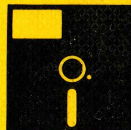
Klein

mit
Programm-
Diskette



# Amiga: Programmieren in Basic

Der sichere Weg zu einer modernen Basic-Variante

Amiga- Einführung
Programmieren in Amiga- Basic
Pie- Charts
3D- Säulengrafik
Menü- Technik
Windows — Fenster im Bildschirm
Screen — Auflösung und Farbe neu definiert
Sprites und Bewegung
Mondlandung mit Fähre
Töne und Geräusche
Ein Programm zum Vorlesen von Listings

Franzis'

Klein
Amiga: Programmieren in Basic

Rolf-Dieter Klein

# Amiga: Programmieren in Basic

Der sichere Weg zu einer modernen Basic-Variante

Mit 81 Abbildungen und 4 Farbtafeln

Franzis'

CIP-Kurztitelaufnahme der Deutschen Bibliothek

**Klein, Rolf-Dieter:**
Amiga: Programmieren in Basic: d. sichere Weg zu e. modernen Basic-Variante / Rolf-Dieter Klein. – München: Franzis, 1987.
ISBN 3-7723-8971-6



Druck: Kösel, Kempten
Printed in Germany · Imprimé en Allemagne.

ISB N 3-7723-8971-6

# Vorwort

Der Amiga ist ein Computer, der schon seit einiger Zeit von sich reden macht. Er glänzt durch seine hervoragenden Grafikfähigkeiten. Doch erst seit kurzer Zeit ist er auch preislich erschwinglich geworden und in den Homecomputer-Markt eingedrungen.
Gerade der Anfänger mußte sich bisher durch eine Fülle von Literatur hindurcharbeiten.
Ziel dieses Buches ist es, dem Anfänger eine Einführung in die Programmierung des Amiga-Basics zu bieten. Dem Umsteiger wird ein Werk angeboten, mit dem er schnell die Besonderheiten des Amiga-Basic nutzen lernt. Dabei wird im Buch besonderer Wert auf eine systematische Einführung gelegt, die gerade die moderne Überarbeitung der Sprache Basic, wie sie im Amiga Verwendung findet, besonders berücksichtigt.

Rolf-Dieter Klein, München

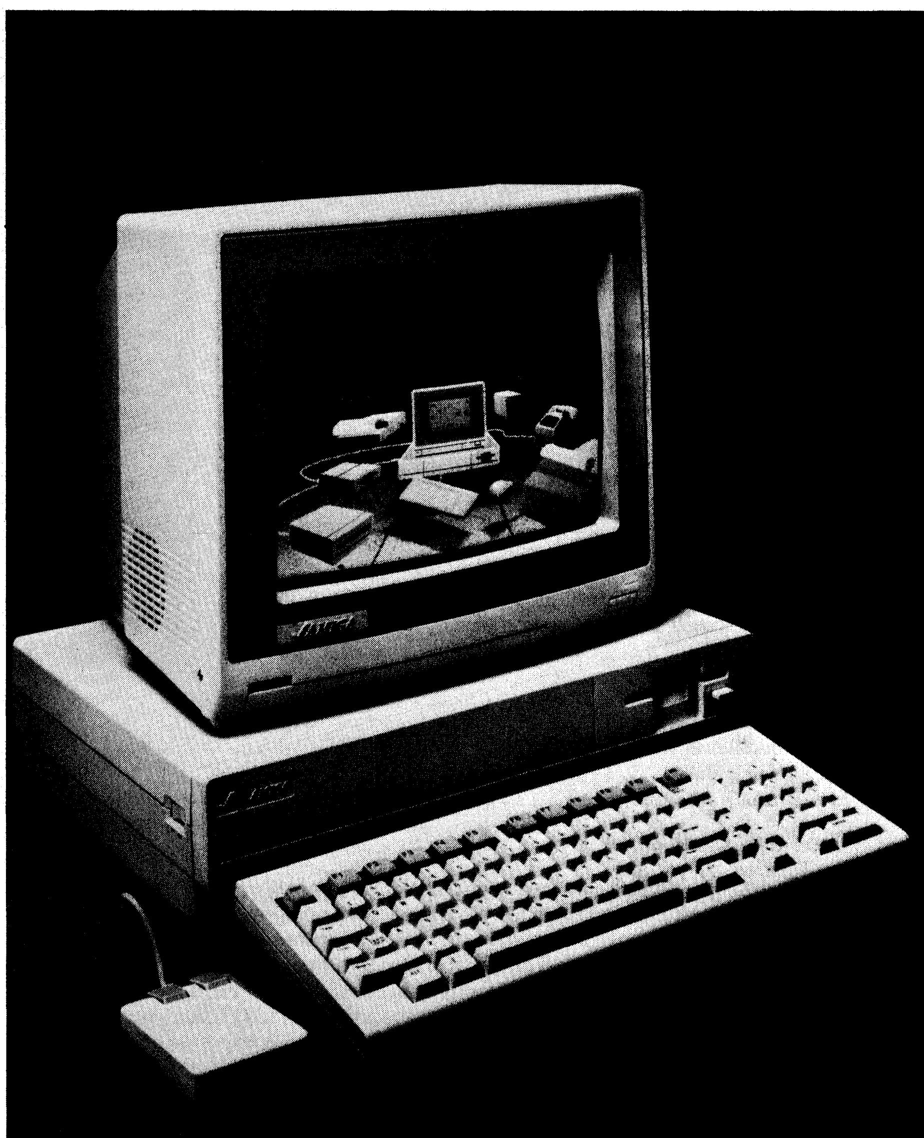
## Wichtiger Hinweis

Die in diesem Buch wiedergegebenen Schaltungen und Verfahren werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden*).
Alle Schaltungen und technischen Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag und der Autor sehen sich deshalb gezwungen, darauf hinzuweisen, daß sie weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung eventueller Fehler sind Autor und Verlag jederzeit dankbar.

---

*) Bei gewerblicher Nutzung ist vorher die Genehmigung des möglichen Lizenzinhabers einzuholen.

# Inhalt



# 1 Amiga-Einführung

Der Amiga ist eine besondere Art von neuem Computer. Er gehört zu den sogenannten 16-Bit-Computern. Das bedeutet im Unterschied zu den sogenannten 8-Bit-Computern eine höhere Rechenleistung und größeren Speicher (beim Amiga können zusätzlich 8 MByte Speicher zugebaut werden). Als Zentraleinheit, gewissermaßen das Herz des Computers, wird ein moderner Baustein mit dem Namen 68000 verwendet. Dieser Baustein gehört zur neuen Generation der Mikroprozessoren und kann ausserordentliches leisten. Neben diesem Standart-Baustein verwendet der Amiga aber auch Bausteine, die es nur im Amiga gibt. Sie werden Agnus, Denise und Paula genannt. Erst durch diese drei Bausteine wird aus dem 68000-Computer ein Amiga. Der 68000 ist allein schon ein schneller Rechner. Die Zusatzbausteine machen ihn aber noch leistungsfähiger. So zum Beispiel bei grafischen Darstellungen. Das Bild entsteht schneller als es der 68000 allein könnte, da der Baustein Agnus die Möglichkeit hat, Bildteile schnell zu manipulieren. Der Baustein Denise hat die Aufgabe, das Bild für den Bildschirm aufzubereiten und bestimmt auch die Farbwahl. Denise kann außerdem sogenannte grafische Objekte (siehe Kapitel Sprites) auf dem Bildschirm hin- und herbewegen. Der Baustein Paula schließlich ist zum Beispiel für die Erzeugung von Ton zuständig. Dadurch sind beim Amiga vier unabhängige Tonkanäle möglich.
Der Amiga besitzt aber auch eine offene Struktur, das heißt, alle wichtigen Funktionen und Programmteile sind ausführlich dokumentiert und jedermann zugänglich, und der Amiga kann erweitert werden. Über eine Buchse im Gehäuse kann man nämlich extern weitere Geräte und Schaltungen anschließen, so weitere Floppy-Laufwerke, Festplatten-Laufwerke oder Analog-Digital-Umsetzer, Relais usw.
Eine weitere Besonderheit ist die Maussteuerung, eine moderne und benutzerfreundliche Art der Eingabe. Die Maus ist ein kleines Kästchen, in dem sich eine Kugel befindet. Wenn man die Maus auf einer Unterlage verschiebt, so dreht sich die Kugel entsprechend der Bewegung. Eine Elektronik nimmt diese Bewegung auf und leitet sie an den Rechner weiter. Der kann aus der Rollbewegung nun auf die Bewegung der Maus zurückschließen. Der Rechner bewegt dann einen Zeiger, zum Beispiel einen Pfeil, entsprechend der Bewegung der Maus auf dem Bildschirm mit. Auf der Maus befinden sich beim Amiga zwei Tasten, mit denen Funktionen ausgeführt werden können. Wenn man mit dem Pfeil zum Beispiel auf ein Programm zeigt und die linke Maustaste drückt, so wird dieses dunkel markiert, drückt man nochmals gleich darauf, so wird das ausgewählte Programm gestartet. Die einzelnen Programme werden als grafische Symbole mit Textnamen auf dem Bildschirm ausgegeben, wie in *Abb.1.1* gezeigt.
Durch die Maustechnik wird vermieden, daß man sich umfangreiche Befehle merken muß, die man bei anderen Systemen über Tastatur eingeben muß.

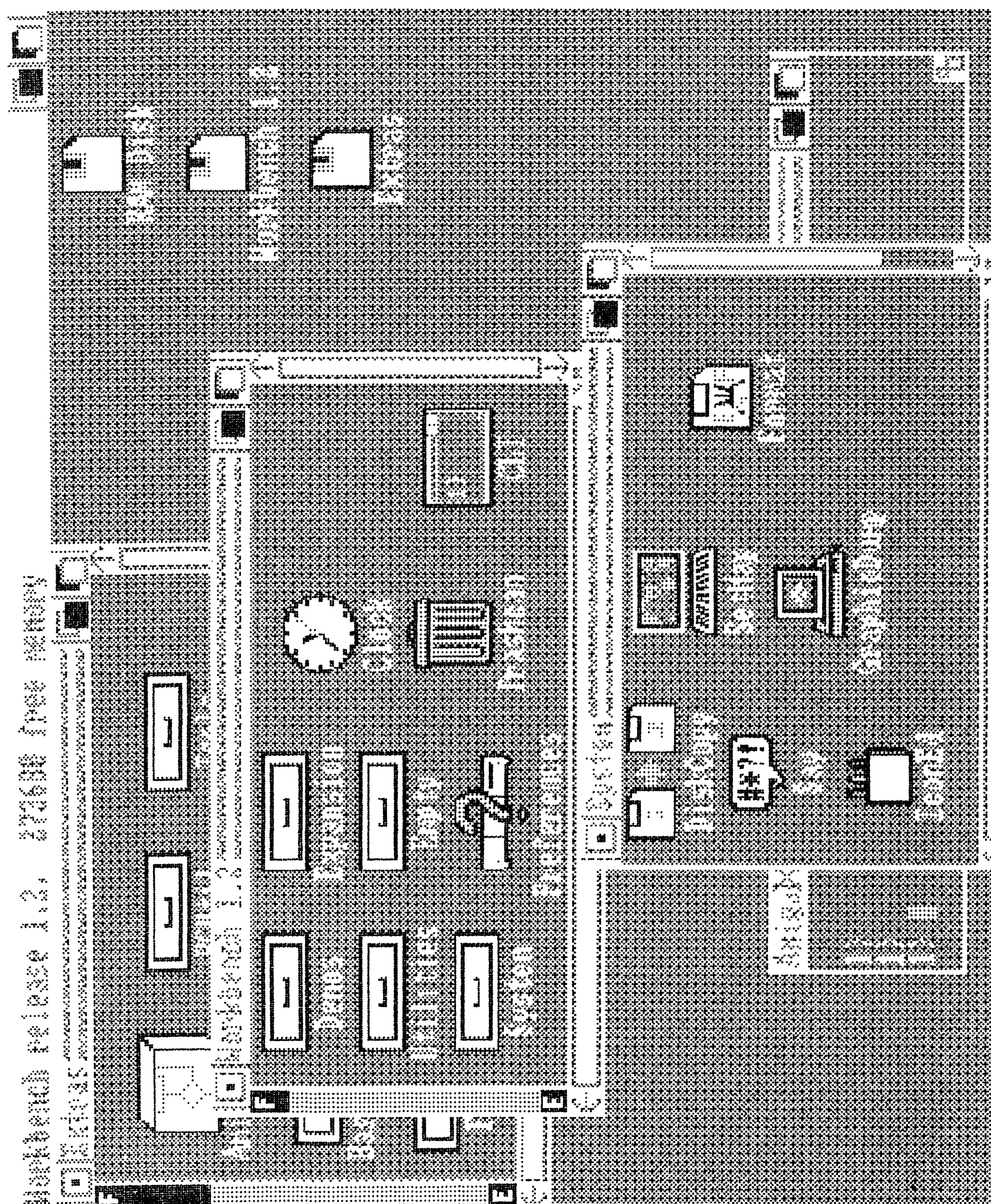


Abb. 1.1 Die Workbench des Amiga

Der Vorteil dieser Maustechnik besteht dann, daß man auch ganz unbedarft an einen solchen Rechner herangehen kann und ihn schon bedienen kann, ohne über alle Befehle Bescheid wissen zu müssen.
Mit der Maus kann man aber nicht nur Programme starten, sondern auch Symbole bewegen oder zeichnen. Man hat damit ein sehr mächtiges Werkzeug. Wie man selbst Programme mit Maussteuerung schreibt, werden Sie in späteren Kapiteln noch lernen.

# 2 Programmieren in Amiga-Basic

Das Amiga-Basic wird auf einer Diskette mit dem Namen "Extras" geliefert. Nachdem man die Disketten Kickstart (nur beim Amiga 1000 nötig) und dann Workbench eingelegt hat, kann man die Diskette Extras anstelle von Workbench einlegen und durch zweimaliges Anklicken des Diskettensymbols aufrufen. Wer die grundsätzliche Bedienung des Amigas noch nicht genau kennt, kann dies auch in den mitgelieferten Handbüchern des Amigas genau nachlesen.
Man sollte übrigens nie die Originaldisketten verwenden, sondern immer nur eine Kopie davon, sonst läuft man Gefahr sie bei Fehlbedienung zu zerstören.
Nachdem man das Diskettensymbol Extras zweimal angeklickt hat, erscheint ein Bild ähnlich zu *Abb 2.1*. Man sieht den Inhalt der Extras-Diskette in grafischer Form.
Links oben ist auch das Programm Amiga-Basic zu sehen. Will man es starten, so positioniert man die Maus über dem Symbol und klickt zweimal (linke Maustaste). Das Basic wird geladen und gestartet. Es erscheint *Abb. 2.2* und das Basic meldet sich mit einem "OK" arbeitsbereit. Sollte das OK nicht erscheinen, so klickt man kurz in den linken Bildteil.

## 2.1 Der Basic-Interpreter - Aufbau und Wirkungsweise

Bevor es so richtig ans Programmieren geht, machen wir einen kleinen Ausflug in die Theorie.
Basic ist eine Programmiersprache, die 1964 auf dem Dartmouth College in New Hamshire (USA) entwickelt wurde. Basic steht als Abkürzung für "Beginners All Purpose Symbolic Instruction Code".
Zu dieser Zeit gab es nur wenige Sprachen zur Programmierung und diese waren alle recht kompliziert für Anfänger. Man suchte nach einem Weg einen schnellen Einstieg in die Programmierung zu ermöglichen.
Was unterscheidet Basic von anderen Programmiersprachen, die damals existierten?

1. Es gibt eine überschaubare Zahl von Befehlen.

2. Man kann Befehle im sogenannten Direkt-Mode sofort ausprobieren, ohne lange warten zu müssen.

Hinzu kommen heute:
3. Sehr große Verbreitung der Sprache Basic. Man findet sie auf fast allen Computersystemen.

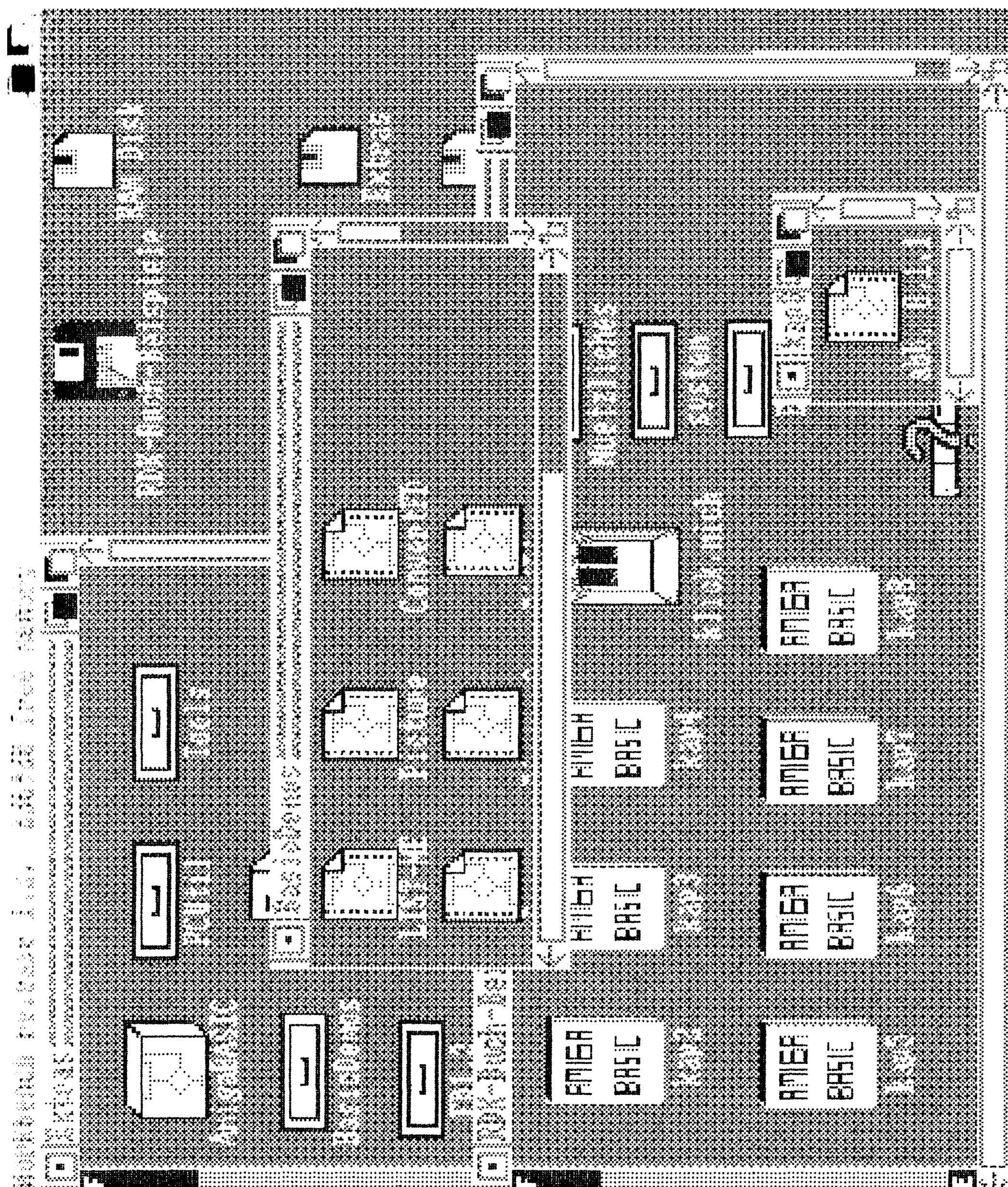


Abb. 2.1 Die Workbench 1.2

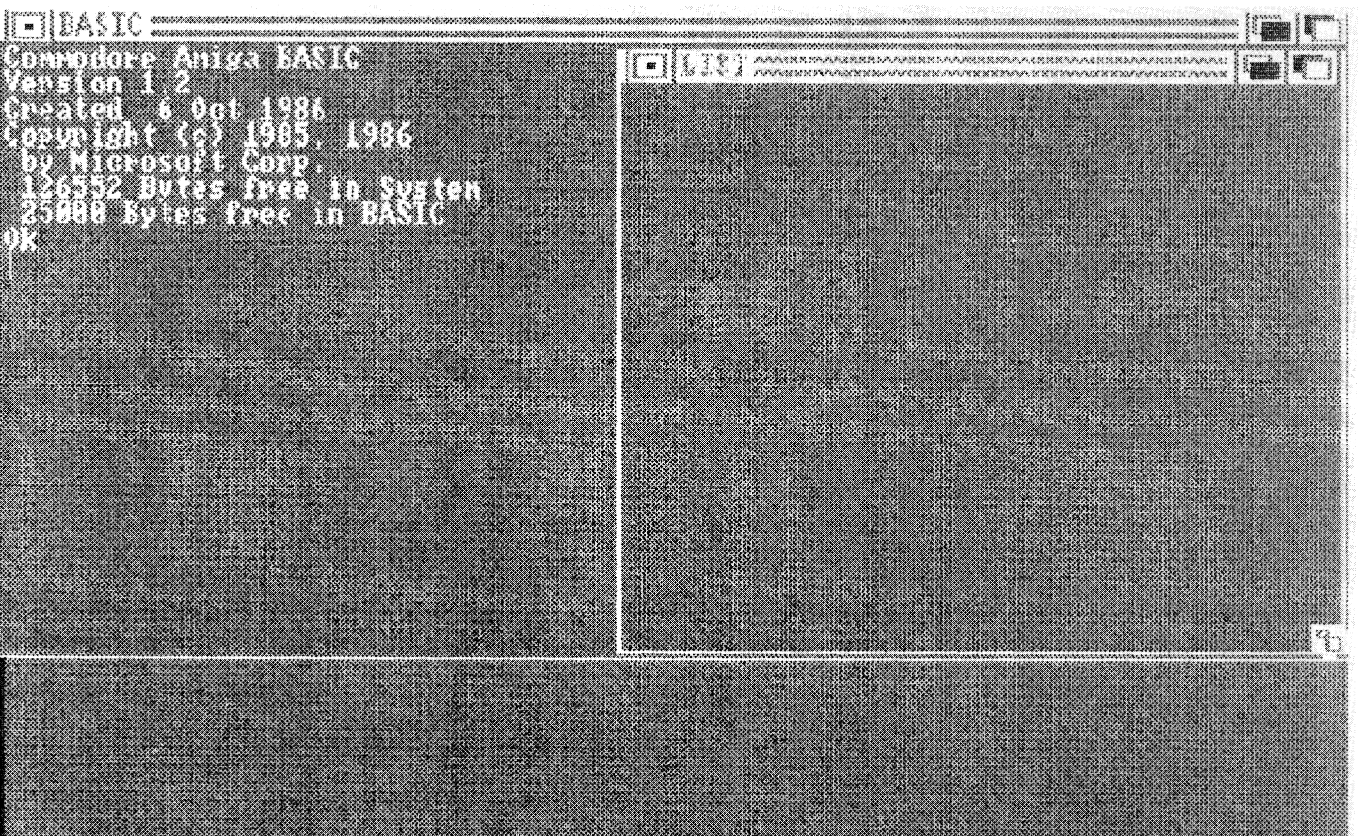



Abb. 2.2 So meldet sich Basic

Man soll aber auch die Nachteile nicht verschweigen.
1. Es gibt eine Vielzahl von Dialekten. Nicht jedes Basic versteht die gleichen Befehle. Dies wurde zwar durch einen Quasi-Standard der Firma Mikrosoft stark verbessert, zählt aber heute noch zu den größten Nachteilen von Basic.
2. Große Programme sind schwer lesbar. Auch dies gilt heute nicht mehr ganz, da neuere Basic-Varianten, wie z.B. das Amiga-Basic, Befehle besitzen, mit denen die Lesbarkeit erhöht wird.

Unterschiedliche Programmiersprachen wurden aber auch entwickelt um unterschiedlichen Aufgaben gerecht zu werden.

Beispiele für Programmiersprachen:

Ada: Eine sehr neue Sprache. Sie vereinigt gewissermassen vielerlei Programmiersprachen in sich und ist daher sehr komplex. Die Sprache ist wegen des Umfangs nur auf wenigen Mikrorechnern verfügbar.

APL: Die Programmiersprache verwendet spezielle mathematische Symbole zur Programmierung und kann daher sehr kompakt Programme schreiben.
Die Sprache ist vorwiegend für mathematische Probleme geeignet.

Algol: Eine Universal-Programmiersprache, die gerne an Universitäten verwendet wurde. Sie besitzt Strukturelemente.

Forth: Eine recht neue Sprache, die die umgekehrt polnische Notation verwendet. Die Sprache wurde ursprünglich zu Steuerungszwecken für die Astronomie entwickelt.

Fortran: Eine Sprache für technisch wissenschaftliche Aufgabenstellungen. Mit dieser Sprache können zum Beispiel komplexe Zahlen direkt verarbeitet werden.

C: Eine neuere Programmiersprache, die sehr schnell ablaufende Programme liefert, allerdings auch Probleme bei der Fehlersuche in sich birgt.

Lisp: Listen orientierte Programmiersprache. Sie wird vorwiegend für das Programmieren von "Künstlicher Intelligenz" verwendet.
Die Sprache unterscheidet sich stark von anderen Programmiersprachen.

Logo: Ebenfalls Listenorientiert wie Lisp, jedoch enthält die Sprache zusätzliche Befehle für die Grafik-Programmierung (Schild-Kröten-Geometrie). Die Sprache wurde ursprünglich für das Programmierenlernen für Kinder entwickelt. Die Sprache kann jedoch auch sehr komplizierte Probleme lösen helfen.

Modula2: Eine Weiterentwicklung der Sprache Pascal. Man kann damit sehr bequem programmieren. Die Sprache eignet sich auch für große Programme sehr gut.

Pascal: Eine Universal-Programmiersprache, ähnlich zu Algol, die Strukturierungs-Elemente besitzt und insbesondere durch Turbo-Pascal verbreitet wurde.

Prolog: Das Programmieren wird hier durch Regeln ermöglicht.
Die Sprache wird ähnlich wie Lisp für "Künstliche Intelligenz"-Aufgaben verwendet, unterscheidet sich jedoch auch von LISP sehr stark.

Damit sind längst nicht alle Programmiersprachen genannt, jedoch ein guter Querschnitt. Die Sprachen Logo, Forth und Lisp erlauben es ähnlich zu Basic auch Programme und Befehle unmittelbar am Bildschirm auszuprobieren.
Neben den höheren Programmiersprachen gibt es auch die sogenannten Assembler. Ein Assembler ist immer speziell für einen bestimmten Computer geeignet. So gibt es Assembler für den Z80, für den 8086, für den 68000 usw. Man kann Programme, die in Assembler geschrieben sind, nicht ohne weiteres auf verschiedenen Computern laufen lassen.
Der Assembler versteht die Maschinensprache des Computers und macht daraus einen Maschinencode. Der Vorteil beim Programmieren in Assembler liegt darin, daß die Programme sehr schnell ablaufen, denn ein Maschinenbefehl wird zu einem Maschinencode übersetzt.
Bei den höheren Programmiersprachen, wie Basic ist das anders.
Man unterscheidet dabei noch zwei Formen:
Den Compiler und den Interpreter.
Der Compiler macht aus einem Programm einen Maschinencode. Dabei muß ein Befehl der Programmiersprache nicht unbedingt in nur einen Maschinencode umgesetzt werden, es können mehrere Maschinencodes erzeugt werden.
Der Interpreter arbeitet noch mal anders. Er erzeugt keinen Maschinencode. Er holt sich die Befehle aus dem Speicher und führt sie dann jeweils gleich aus. Dabei muß er zunächst herausfinden, welcher Befehl es ist und was er bedeutet. Dieses anschauen tut er auch, wenn er wieder zum gleichen Befehl kommt und das ist sehr zeitaufwendig. Interpreter sind daher ca. 10 bis 100mal langsamer als vergleichbare Compiler, was die Ausführungszeit der Programme angeht. Der Vorteil vom Interpreter liegt darin, daß er sofort anfängt das Programm auszuführen, wenn man es eingetippt hat. Der Compiler hingegen muß zunächst das gesamte Programm in Maschinencode übersetzen und dann wird es ausgeführt. Das Übersetzen kann einige Minuten dauern. Wenn man das Programm aber nicht mehr ändert, so muß es beim Compiler auch nicht neu übersetzt werden, man verwendet einfach den fertigen Maschinencode und das Programm beginnt sofort und schnell seine Arbeit. Auch ein gemischtes Vorgehen ist möglich. Man entwickelt das Programm mit einem Interpreter. Wenn dann alles fertig ist, "compiliert" man das Programm und hat ein schnell ablaufendes fertiges Produkt.
Wir wollen uns im Verlauf des Buchs mit dem Interpreter "Amiga-Basic" befassen, da gerade der Anfänger von den Vorteilen eines Interpreters profitiert.

## 2.2 Direkt-Befehle - einfache Beispiele

Fangen wir doch gleich mit dem Programmieren an. Das Amiga-Basic arbeitet mit der sogenannten Fenster-Technik. Zwei Fenster sind in Abb 2.2 sichtbar. Links das Basic-Fenster und rechts das List-Fenster. Wenn man nun einen Buchstaben auf der Tastatur tippt, so erscheint er in einem der beiden Fenster. Wo, das hängt davon ab,

wo gerade der Cursor blinkt. Dies ist beim Amiga-Basic eine senkrechte blinkende Linie. Wenn Sie mit der Maus in das linke Fenster gehen und dort einmal anklicken, so blinkt der Cursor anschließend dort. Führen Sie den Mauszeiger in das rechte Fenster und klicken dort, so blinkt dort ein Cursor.
Experimentieren Sie einmal damit.
Nun tippen Sie in das linke Fenster den Text:

**PRINT 2*3**

Wenn Sie die Zeile eingegeben haben, drücken Sie anschließend die RETURN-Taste. Erst dann wird der Befehl ausgeführt. Als Ergebnis erscheint auf dem Bildschirm der Wert 6. Wenn Sie sich vertippt haben, so erscheint z.B. die Meldung "SYNTAX ERROR" oder die Meldung "UNDEFINED SUBPROGRAM" im oberen Bildteil. Der Basic-Interpreter signalisiert damit, daß er den Befehl nicht verstanden hat. Um danach weitermachen zu können, müssen Sie den Mauszeiger in das "OK"-Feld bringen und einmal anklicken (linke Maustaste).
Danach müssen Sie die Zeile nochmals korrekt eintippen. Tippfehler kann man vor Eingabe der RETURN-Taste auch durch Betätigen der Backspace (Pfeil nach links, siehe Commodore-Anleitung) beheben.
Später lernen wir mit dem List-Fenster umzugehen, denn dort geht das nach Auftreten eines Fehlers bequemer.
Das Basic unterscheidet nämlich zwei Betriebsarten.

1. Den Direkt-Mode
2. Den Programm-Mode.

Wenn Sie Befehle in das Basic-Fenster eintippen, werden diese immer im Direkt-Mode ausgeführt. Tippen Sie die Befehle in das List-Fenster, werden sie als Programm gespeichert. Befehle, die man dort eingibt, werden nicht gleich ausgeführt, sondern erst gesammelt.
Der Begriff SYNTAX ist vorher schon einmal gefallen. Was bedeutet er?
Unter der Syntax versteht man die erlaubte Zeichenfolge in einer Sprache. Es ist dies die Grammatik.
Der Basic-Interpreter kann Fehler dieser Art selbstständig erkennen und uns darauf aufmerksam machen. Er kennt die Syntax der Sprache Basic und vergleicht eingegebene Befehle mit der Syntax.
Nicht erkennen kann er sogenannte semantische Fehler. Dies sind Fehler, die der Bedeutung, also dem Sprachinhalt entsprechen.
Beispiel:
Wenn Sie vorher versehentlich **PRINT 2+3** eingetippt haben, so rechnet der Computer brav das Ergebnis 5 aus. Sie wollten aber eine Multiplikation durchführen und haben nur das Zeichen "+" mit dem Zeichen "*" verwechselt. Da aber beide Zeichen in diesem Zusammenhang erlaubt sind, kann der Basic-Interpreter keine Fehlermeldung bringen.
Es gibt viele solcher Beispiele, wie man Fehler macht, die der Basic-Interpreter nicht erkennen kann. Leider sind die meisten dieser Beispiele nicht so leicht zu durchschauen. Solche Fehler zu finden gehört zur hohen Kunst des Programmierens. Als Anfänger macht man aber auch sehr häufig syntaktische Fehler und Sie werden die Fehlermeldung "SYNTAX ERROR" oder "UNDEFINED SUBPROGRAM" öfters bekommen als Ihnen lieb ist. Vergleichen Sie dann sorgfältig das im Buch abgedruckte Programm mit dem eingebenen. Manchmal übersieht man auch gerne sogenannte Leerzeichen. Beispiel "A B" oder

"AB". Leerzeichen gibt man wie bei einer Schreibmaschine mit der großen langen Taste ein.
Versuchen wir doch mal einen Syntax-Fehler zu bekommen. Tippen Sie dazu ein:

```
PRINT =
```

Die Meldung "SYNTAX ERROR" erscheint links oben. Danach muß man "OK" anklicken. Auch die andere Fehlermeldung läßt sich erzeugen. Tippen Sie:

```
PRUNT 3+2
```

Die Meldung "UNDEFINED SUBPROGRAM" kommt prompt auf dem Bildschirm. Sie müssen dann "OK" anklicken, um weiter machen zu können. Hier heißt die Meldung nicht "SYNTAX ERROR", da der Basic-Interpreter meint, Sie wollten einen neuen Befehl verwenden. Das Amiga-Basic erlaubt es nämlich mit bestimmten Einschränkungen, eigene neue Befehle hinzuzufügen. Andere Basic-Interpreter melden hier nur "SYNTAX ERROR".
Mit Fehlern richtig umgehen zu können ist eines der wichtigsten Ziele, wenn man Programmieren lernen will. Man muß lernen Fehler zu erkennen, man muß sie einplanen und natürlich beseitigen.
Doch fangen wir erst mal mit der Syntax an.
In Basic werden grundsätzlich zwei verschiedene Zahlenarten unterschieden:
Die sogenannten Integer-Zahlen, was soviel wie Ganze Zahlen bedeutet und die Gleitkommazahlen.
Eine Integerzahl besitzt keine Nachkommastellen und auch keinen Exponenten. Gültige Integerzahlen sind: 32000, 0, -23 usw.
Eine Gleitkommazahl kann Nachkommastellen besitzen, aber auch einen Exponenten.
In Basic sind dies zum Beispiel: -1.234 , 3E7, -1.1E-32, 0.00001 usw.
Dabei ist der Zahlenbereich aber auch eingeschränkt. Integerzahlen liegen zwischen -32768 und +32768 und Gleitkommazahlen haben einen Exponenten von -38 bis +38, wobei bis zu 7 Stellen genau gerechnet wird. Der Exponent wird stets durch das Zeichen "E" eingeleitet, Nachkommastellen werden durch einen Punkt (nicht Komma) gekennzeichnet.
Integerzahlen haben den Vorteil, daß sie weniger Speicherplatz brauchen und Rechnungen mit ihnen schneller ausgeführt werden können.
Wem die Genauigkeit nicht ausreicht, der kann auch noch doppelt genaue Zahlen angeben, indem anstelle des "E" ein "D" angegeben wird, doch wir begnügen uns fürs erste mit den Standardeinstellungen.
Beispielsweise geben Sie den folgenden Befehl ein:

**PRINT 2E-5**

auf dem Bildschirm erscheint dann:

**.00002**

Der Basic-Interpreter gibt Zahlen mit mehrdeutiger Schreibweise immer mit einer bestimmten voreingestellten Schreibweise aus. Sie können das Schema leicht anhand einiger Übungen selbst herausfinden.

Natürlich will man auch rechnen. Dazu gibt es zunächst einmal die Grundrechenarten.
Addieren geschieht mit dem Zeichen "+", subtrahieren mit dem Zeichen "-", multiplizieren mit dem Zeichen "*" und die Division wird mit dem Zeichen "/" durchgeführt. Will man die Rechnungen ineinander verschachtelt, so verwendet

man dazu die Klammern "(" und ")".
Beispiel:

**PRINT 3*(4+7.2*(112E3-111E3)/(4.3 - 2.2))**

liefert das Ergebnis:

**10297.71**

Man kann dabei auch Leerzeichen zwischen den einzelnen Operationen verwenden, nicht jedoch innerhalb einer Zahl. Auch die Schreibweise "e" oder "E" beim Exponenten spielt keine Rolle.
In der Mathematik werden Formeln meist anders geschrieben, z.B.:

$$\frac{3.4 + 7.2 * (6 + 2)}{3.4 - 2.1*3}$$

Diese Schreibweise muß man dann mit Hilfe von Klammern in die Rechner-Schreibweise umwandeln:

(3.4 + 7.2 * (6 + 2)) /(3.4 - 2.1*3)

Weitere Verknüpfungen sind z.B. "^", womit man die Potenzierung durchführen kann.
Beispiel:

**PRINT 3^3.2**

ergibt

**33.63474**

Eine spezielle Form der Division wird durch das Zeichen "\" dargestellt. Es ergibt sich nur der ganzzahlige Anteil der Division.
Beispiel:

**PRINT 13\3**

ergibt

**4**

anstelle von 4.333333 wie man es bei "/" erhält.
Dazugehörig ist auch die Operation "MOD", wobei man hier nicht ein einzelnes Zeichen verwendet, sondern ein ganzes Wort. MOD steht für Modulo und gemeint ist die Restfunktion.
Beispiel:

**PRINT 13 MOD 3**

ergibt

**1**

da 13 geteilt durch 3 als Ergebnis 4 Rest 1 hat.
Anwendungen für die Restfunktion werden wir später noch sehen.

Neben den Grundrechenarten gibt es aber auch mathematische Funktionen. Erschrecken Sie dabei nicht, wenn Sie die Funktionen nicht vom Mathematikunterricht kennen, man braucht sie nicht gleich alle kennenzulernen. Funktionen werden in Basic durch einen Namen angegeben, hinter dem in Klammern die Werte stehen.

**ABS(zahl)**
Absolutwert bilden. ABS(-3.2) liefert das Ergebnis 3.2.

**ATN(zahl)**
Arcustangens bilden. Das Ergebnis wird in Radiant ausgeben.

Beispiel:

**PRINT ATN(10000)**

liefert

**1.570696**

**COS(zahl)**
Cosinus eines Wertes, der in Radiant angegeben wird.
COS(3.141592) ergibt -1.

**EXP(zahl)**
Bilden der e-Funktion. EXP(1) ergibt 2.718282.

**SIN(zahl)**
Sinus eines Wertes, der in Radiant angegeben wird.
SIN(1.2) ergibt .9320391.

**SQR(zahl)**
Quadratwurzel ziehen. Der Wert von "zahl" muß dabei positiv sein, sonst erfolgt eine Fehlermeldung (Illegal function call). Beispiel:

**PRINT SQR(2)**

ergibt

**1.414214**

**TAN(zahl)**
Tangens einer Zahl, die in Radiant angegeben wird. TAN(1.5) ergibt 14.10142.

**LOG(zahl)**
Der Logarithmus zur Basis e wird gebildet. TAN(2.71828) ergibt 0.9999993.

## 2.3 Das erste Programm - Eingabe und Test

Bisher hatten wir den Computer nur als teueren Taschenrechner verwendet. Das soll jetzt anders werden. Es geht ans Programmieren.
Dazu verwendet man beim Amiga das Fenster mit der Beschriftung "List". Sollte es nicht sichtbar sein, so kann man es auch durch Drücken der rechten Amigataste "A" gleichzeitig mit der Taste "L" wieder erscheinen lassen.
Das geht auch, wenn man die rechte Maustaste drückt und den Menüpunkt (ganz oben mit Basic-Window) "Windows" und dort das Unterfeld "Show List" anwählt. Wie man Menüs anwählt, sollte man einmal in den Amiga-Handbüchern nachlesen. Klicken Sie mit der linken Maustaste einmal in das "List"-Fenster. Dann kann man dort hineinschreiben. Geben Sie den folgenden Befehl dort ein:

**PRINT 3*4**

Wenn Sie am Schluß der Zeile die Return-Taste gedrückt haben, so wird hier nicht mehr der Wert 3*4, also 12 ausgegeben, sondern der Cursor springt in die nächste Zeile. Ferner wird "print", falls Sie es mit Kleinbuchstaben eingegeben haben, automatisch in Großbuchstaben dargestellt, um zu kennzeichnen, daß es sich um einen Befehl handelt.

Will man dieses einfachste Programm starten, so gibt es mehrere Möglichkeiten.
1. Drücken Sie die Amiga-Taste zusammen mit der "R"-Taste. Links im "Basic"-Fenster erscheint das Ergebnis 12.
2. Klicken Sie das "Basic"-Fenster an und tippen Sie dann den Befehl "RUN" ein. Das Programm wird gestartet und auch wieder der Wert 12 ausgegeben.
Will man jetzt das möglicherweise verschwundene "List"-Fenster wieder sichtbar machen, so genügt es z.B. den Befehl LIST einzugeben.
Nun kann man das Programm weiter ergänzen. Dazu muß man vorher wieder das "List"-Fenster anklicken, um einen Cursor zu bekommen.
Innerhalb des "List"-Fensters kann man sich auch mit den Cursor-Tasten (Pfeil nach oben, links, rechts und nach unten) bewegen, aber nur auf schon definiertem Gebiet, sonst blinkt der Schirm kurz hell auf. Man kann auch die Maus zum Positionieren des Cursors verwenden, indem man den Mauszeiger an die neue Position führt und dann mit der linken Maustaste anklickt.
Positionieren Sie den Cursor in die zweite Zeile. Dort geben Sie die folgenden Programmstücke ein:

**PRINT "Hallo es geht"**
**PRINT "Rechnung: sin(0.4) = ";SIN(.4)**

Wenn Sie danach die Tasten "Amiga" und "R" drücken, erhalten Sie *Abb.2.3.1*. Dort ist auch das vollständige Programm im "List"-Fenster zu sehen.
Man kann das Programm natürlich auch mit "RUN" starten.
Jede Programmzeile wird dabei nacheinander abgearbeitet. Dabei beginnt der Basic-Interpreter immer mit der ersten Programmzeile. Dann holt er sich die zweite Zeile, dann die dritte, bis keine Programmzeilen mehr vorhanden sind. Nun wird die Programmausführung beendet. Später werden wir noch Befehle kennenlernen, mit denen man diese sequenzielle Programmausführung ändern kann, um zum Beispiel Wiederholungen zu programmieren.
Der PRINT-Befehl wurde hier auch schon etwas komfortabler verwendet, als am Anfang. So kann man bei der Print-Anweisung auch Texte, die in Anführungszeichen stehen müssen, mit ausgeben, oder man kann mehrere Werte in einer Zeile, durch Semikolon oder Komma getrennt, angeben.
Hier stoßen wir gleich auf eine Schwierigkeit bei Programmiersprachen. Wie definiert man eigentlich die Syntax einer Sprache. Zunächst einmal kann man das rein verbal tun, doch meist ist die Sprache nicht exakt genug dazu. Eine bessere Darstellung kann man in Form von Syntax-Diagrammen durchführen. Dazu zeigt *Abb. 2.3.2* das Syntaxdiagramm für die PRINT-Anweisung.
Die Linien mit den Pfeilen geben dabei die Richtung an, in der man durch das Diagramm hindurchlaufen kann. Man beginnt dabei ganz links und muß nach rechts durchkommen. Zunächst steht in einem ovalen Kasten der Text "PRINT", also muß man an dieser Stelle den Text PRINT eintippen. Dann kann man schon nach rechts weiter gehen und ist fertig. Der Befehl PRINT kann also gemäß Syntaxdiagramm auch alleine stehen. Wenn man PRINT vom Basic-Interpreter ausführen läßt, so wird eine Leerzeile ausgegeben.
Nun kann man im Syntaxdiagramm aber auch andere Abzweigungen einschlagen. In eckigen Kästen steht nicht der Text, den man direkt eingeben kann, sondern hierfür müßte man ein weiteres Syntaxdiagramm angeben.
So steht "Basic-Formel" für die Formeln, wie wir einen Teil schon kennengelernt haben. Mit "Basic-Text" ist eine Vorschrift gemeint, die die Textausgabe

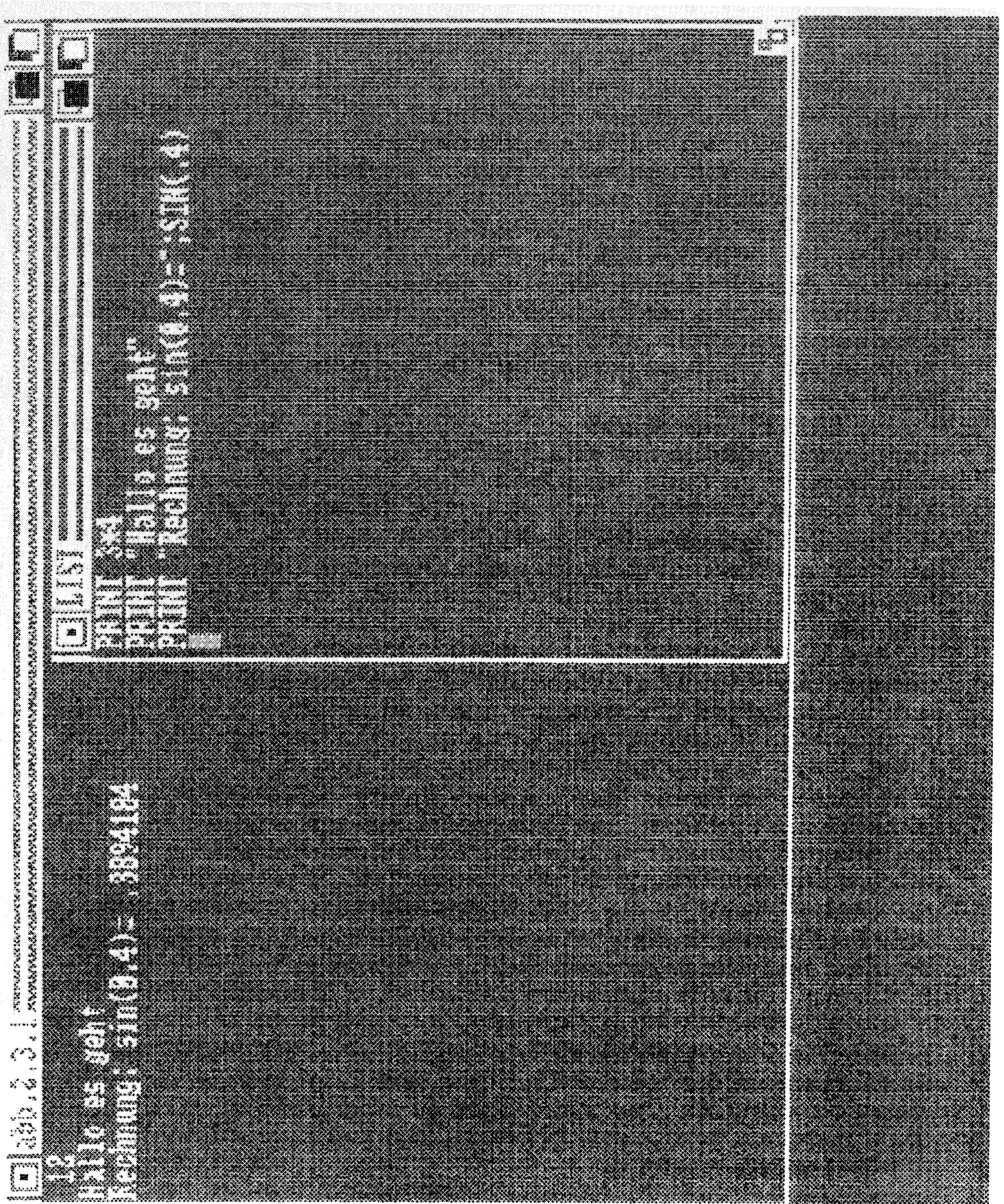



Abb. 2.3.1 Ein kleines Basic-Programm

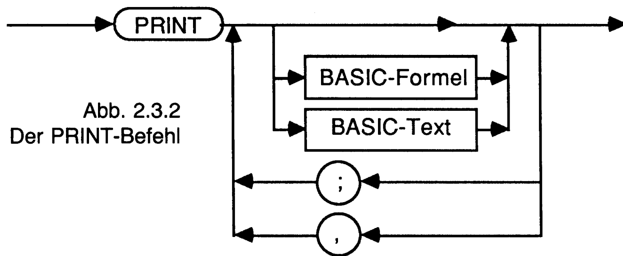



Abb. 2.3.2
Der PRINT-Befehl

ermöglicht. Den einfachsten Fall, einen Text in Anführungszeichen, haben wir vorher schon ausprobiert. Weitere Möglichkeiten folgen noch.
Wenn man mehrere Formeln und Texte hinter dem PRINT-Befehl angeben will, muß man diese voneinander trennen. Dies geschieht entweder durch ein Komma oder durch ein Semikolon.
Beide haben eine unterschiedliche Bedeutung. Wenn man ein Komma verwendet, so wird bei der Ausgabe eine neue Spalte in größerem Abstand verwendet. Bei einem Semikolon wird, wenn es sich um Text handelt, gar kein Leerzeichen als Abstand verwendet und bei Zahlen mindestens ein Leerzeichen.
Beispiel:

**PRINT 1,2,"A";"B";1;2**

führt zu:

**1 2 AB 1 2**

Probieren Sie doch einfach mal selbst ein paar Varianten aus.
Gemäß Syntaxdiagramm ist es aber auch möglich, mit einem Komma oder Semikolon zu enden. Dann wird einfach kein Zeilenvorschub bei der Ausführung des Befehls ausgegeben.
Beispiel (muß als Programm eingegeben werden):

**PRINT "Hallo:";**
**PRINT "Geht"**

führt zu:

**Hallo:Geht**

Damit kann man mehrere Programmzeilen verwenden, um eine Ausgabezeile aufzubereiten.
Wenn Sie übrigens ein Programm löschen wollen, so geben Sie einfach den Befehl **NEW** ein.
Mit **SAVE "beliebigername"** kann man zuvor das Programm auf Diskette abspeichern und mit **LOAD "gleichername"** kann man das Programm auch wieder laden.

## 2.4 Speicherzellen und Namen für Zahlen und Text

Bisher konnten wir noch keine besonders interessanten Programme schreiben. Dazu brauchen wir noch ein paar neue Befehle und Eigenschaften.
Um Zahlen oder Texte abspeichern zu können, gibt es sogenannte Variable. Das sind Speicherzellen im Computer, in die man etwas ablegen kann.
*Abb. 2.4.1* zeigt das Syntaxdiagramm für den Begriff "Zahlvariable". Eine

Zahlvariable:

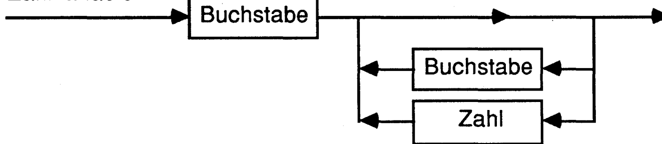



Abb. 2.4.1
Zahlvariable

Textvariable:




Abb. 2.4.2
Textvariable

Zuweisung:

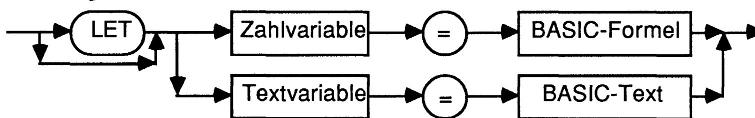



Abb. 2.4.3
Zuweisung

Zahlvariable kann demnach aus einer Kombination aus Buchstaben und Ziffern bestehen. Dabei muß eine solche Variable aber mit einem Buchstaben beginnen.
Beispiele für gültige Zahlvariable:

geht, nunWert, V1, G76C35, H

Nicht gültig sind dagegen:

0456H, Z#1wm, V 1   2,

Man merke: Sonderzeichen oder Leerzeichen dürfen nicht in einer Variable vorkommen.
Neben den Zahlvariablen gibt es aber auch Textvariablen in denen sich Texte speichern lassen. Diese werden zur Kennzeichnung am Schluß zusätzlich mit einem "$"-Zeichen versehen. *Abb. 2.4.2* zeigt das dazugehörige Syntaxdiagramm.
Beispiel für Textvariable:

Alpha$, H$, G5Z665$

Nun nützen die Variable allein aber noch nicht viel, man muß etwas mit ihnen anfangen können, zum Beispiel einen Wert zuweisen. Und dazu zeigt *Abb.2.4.3* das Syntaxdiagramm einer Zuweisung.
Einer Zahlvariablen kann man einen Zahlenwert zuweisen, z.B. das Ergebnis einer Berechnung, und einer Textvariablen einen Text.
Programmbeispiel:

```
A = 3 * 4
PRINT A
```

Das Ergebnis 12 wird auf dem Bildschirm ausgegeben. In A ist der Wert 3 * 4 gespeichert. Oder z.B.:

```
PI = 3.141592
R =  100
PRINT 2 * PI * R
```

Das Ergebnis 628.2919 wird ausgegeben.

Ähnlich kann man mit Textvariablen verfahren. Beispiel:

**ANREDE$ = "Meier"**
**PRINT "Sehr geehrter Herr ";NAME$**

Auf dem Bildschirm erscheint dann:

**Sehr geehrter Herr Meier**

Achtung: Wenn man einen Namen verwendet, der schon als Befehl reserviert ist, so erhält man eine Fehlermeldung. Z.B. NAME$ = "Meier" führt zu einem Syntax-Fehler, da "NAME" schon ein Befehl mit anderer Bedeutung ist. Im Zweifelsfall sollte man im Amiga-Basic-Handbuch nachsehen, ob der Name schon eine Bedeutung besitzt.
Schön wäre es, wenn man die Variable erst nach dem Start eines Programms auf Anfordung einlesen lassen könnte. Dazu gibt es den INPUT-Befehl. *Abb. 2.4.4* zeigt die Syntax dieses Befehls. In *Abb. 2.4.5* ist dazu noch der Teilbegriff "Zeichenkette" definiert.
Beispiel:

**INPUT "Radius:";R**
**PI = 3.141592**
**PRINT "Umfang = ";2*PI*R**

Wenn man das Programm z.B. mit RUN startet, wird der Text:

**Radius:?**

ausgegeben. Dahinter steht der Cursor. Nun kann man einen Wert eintippen und die Return-Taste drücken. Der Umfang wird ausgegeben.
Achtung: Wenn der Bildschirm kurz hell aufblinkt, während man versucht den Wert einzugeben, so muß man mit der linken Maustaste einmal kurz in das "Basic"-Fenster klicken, um es aktiv zu machen. Der Basic-Interpreter weiß sonst nicht,wohin die Eingabe erfolgen soll, denn beim Amiga ist es möglich mit mehreren Fenstern zu arbeiten und dort individuell Eingaben durchzuführen.

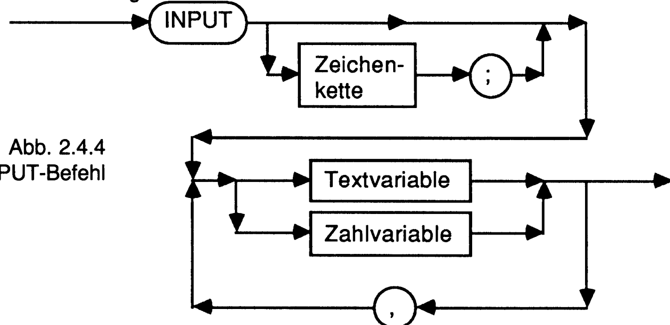



Abb. 2.4.4
Der INPUT-Befehl

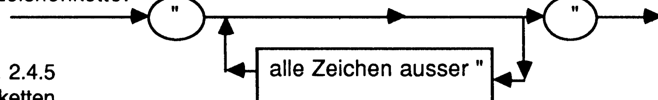



Abb. 2.4.5
Zeichenketten

FOR-Anweisung:

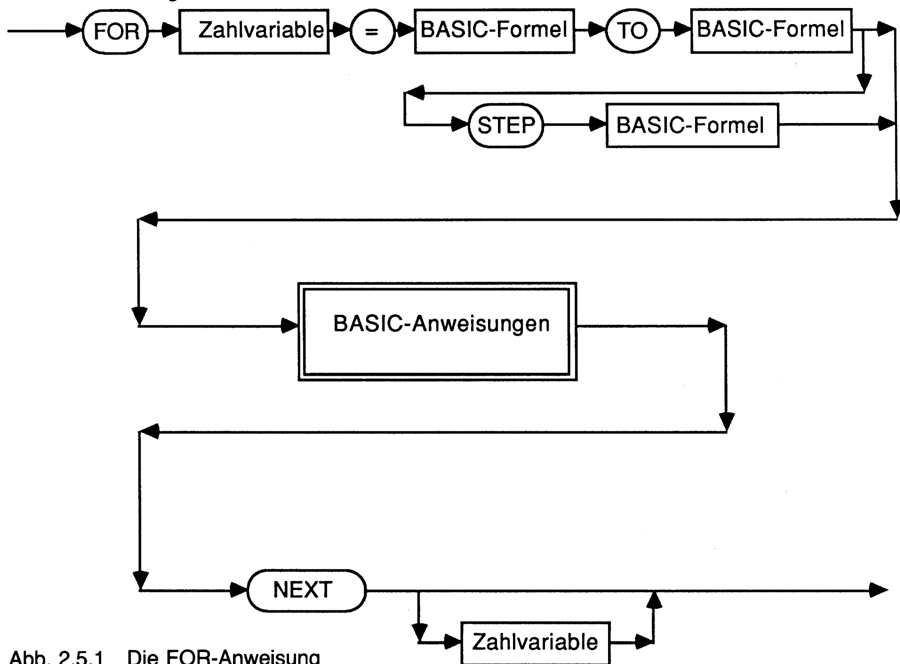



Abb. 2.5.1 Die FOR-Anweisung

## 2.5 Die Wiederholschleife

Man kann nun beliebig Programme dieser Art eintippen und sie werden Schritt für Schritt ausgeführt, bzw. Zeile für Zeile. Interessant ist es aber zum Beispiel, eine Schleife zu bilden, also z.B. eine Formel mit verschiedenen Werten zu durchlaufen. Dazu gibt es die FOR-Anweisung. *Abb. 2.5.1* zeigt das dazugehörige Syntaxdiagramm.

Hinter dem Befehlswort FOR steht als nächstes eine Zahlvariable. Dann folgt ein Gleichheitszeichen, wie bei einer Zuweisung. Dieser Zahlvariable wird nämlich ein Startwert zugewiesen. Danach erfolgt das Wort TO und eine weitere Zahlvariable. Dies ist der Endwert, der für die Zahlvariable genommen werden kann. Danach kann nun das Wort STEP mit einer Wertangabe stehen oder auch nicht. Dies ist die Schrittweite. Gibt man kein STEP an, so wird die Schrittweite 1 angenommen. Danach können z.B. in weiteren Basic-Zeilen beliebige Befehle kommen. Danach kommt die NEXT-Anweisung, die das Ende der Schleife angibt. Man kann wenn man will hier nochmals die Zahlvariable angeben, jedoch ist es nicht unbedingt erforderlich und dient nur der Kontrolle.

Beispiel:

```
FOR i=1 TO 10 STEP 2
  PRINT i,i*i
NEXT i
```

Man erhält eine Tabelle auf dem Bildschirm, die wie folgt aussieht:

| | |
|---|---|
| **1** | **1** |
| **3** | **9** |
| **5** | **25** |
| **7** | **49** |
| **9** | **81** |

Der PRINT-Befehl wird dabei 5 mal wiederholt. Die Variable wird bei Schleifenbeginn mit dem Startwert 1 belegt. Dann wird die Schleife durchlaufen. Bei Auftreten von NEXT springt der Basic-Interpreter zur FOR-Anweisung zurück, diesmal wird aber der Variablenwert um 2 erhöht, falls der Wert 10 noch nicht überschritten wurde, wird die Schleife erneut ausgeführt, das geht dann so weiter bis der Wert 11 erreicht ist, der zum Abbruch der Schleife führt und in der Schleife nicht mehr verwendet wird.
Man kann übrigens auch rückwärts zählen, wenn man einen Startwert angibt, der höher ist als der Endwert und bei STEP eine negative Schrittweite.
Bisher haben wir nur mit Zahlen gearbeitet, doch der Amiga ist für seine Graphik bekannt geworden, daher wollen wir hier mal einen einfachen Grafikbefehl kennenlernen. *Abb. 2.5.2* zeigt das Syntax-Diagramm des Linien-Befehls. Dabei bietet dieser Befehl eine Fülle von Optionen. Zunächst einmal gibt man zwei Punkte der Linie an. Den Startpunkt (x1,y1) und den Endpunkt (x2,y2). Dabei wird ein Koordinatensystem, wie es *Abb. 2.5.3* zeigt, verwendet. Ungewöhnlich ist hier, daß der Nullpunkt links oben liegt, und nicht wie sonst in der Mathematik üblich, links unten. Man hat sich aber bei fast allen Bildschirmsystemen auf dieses Koordinatensystem geeinigt, da es einige Vorteile bringt, wenn man Grafik mit

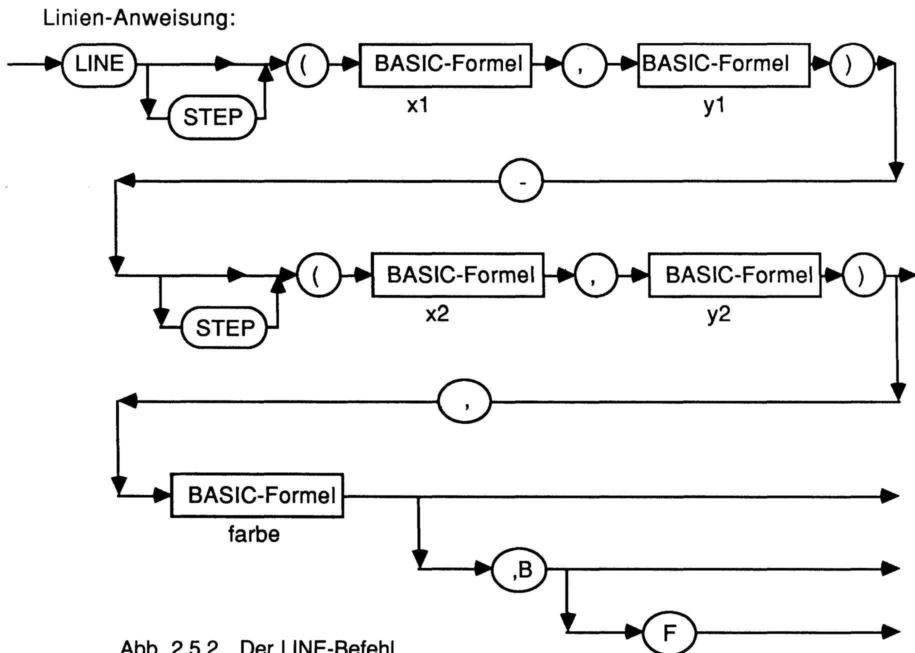



Abb. 2.5.2 Der LINE-Befehl

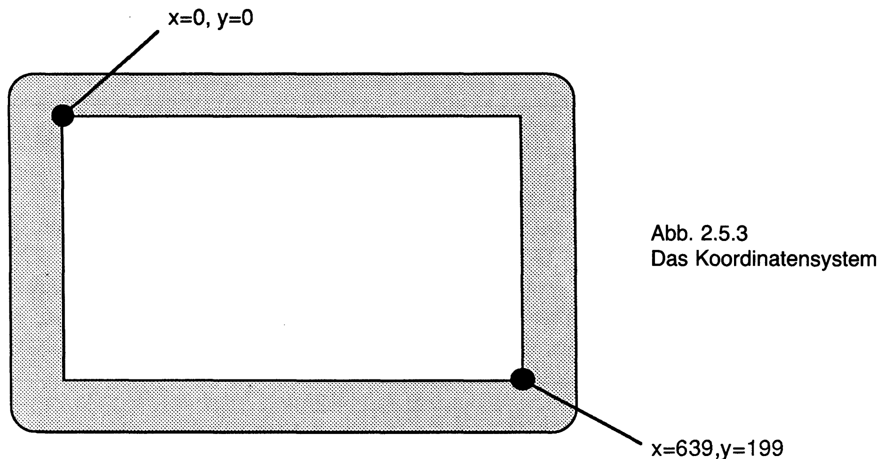



Abb. 2.5.3
Das Koordinatensystem

Text mischt, denn normaler Text wird ja auch von links oben beginnend gelesen. Nach rechts hat man 640 Punkte und daher reicht das Koordinatensystem von x=0 bis x=639. Von oben nach unten hat man 200 Punkte. Wenn man Linien im vollen Bereich darzustellen versucht, so werden normalerweise etwas weniger Punkte vorhanden sein, denn das "Basic"-Fenster besitzt auch einen Rahmen und der braucht natürlich auch Platz. Mit Befehlen, die wir noch kennenlernen, kann man aber auch andere Fensterarten verwenden.
Man kann bei dem Befehl optional auch eine Farbe angeben, in der die Linie gezeichnet wird. Dabei hängt die Anzahl der Farben von der gewählten Bildschirm-Darstellung ab. Beim Standard-Schirm, so wie er nach dem Einschalten vorliegt, sind vier Farben möglich. Welche, das hängt von der individuellen Einstellung durch das Programm PREFERENCES ab, wir gehen einmal von den Standardfarben aus.
Die Farbe mit dem Code 0 ist dabei die Hintergrundfarbe (Blau), der Code 1 ist der Schreibfarbe zugeordnet (Weiß), der Code 2 entspricht normalerweise der Farbe Schwarz und der Code 3 ist der Farbe Rot zugeordnet.
Mit einem kleinen Programm kann man sich z.B. Linien in allen Farben ausgeben:

```
FOR y = 20 TO 100 STEP 5
  farbe = y MOD 4
  LINE (50,y)-(150,y),farbe
NEXT y
```

Auf dem Bildschirm erscheinen waagrechte Linien in unterschiedlichen Farben. Dabei fehlt die Linie mit dem Code 0 immer, denn die Farbe ist ja identisch mit der Hintergrundfarbe und daher nicht sichtbar.
Das Programm ist hier recht trickreich aufgebaut. Der Code für die Variable "farbe" wird mit Hilfe der MOD-Funktion berechnet. y MOD 4 liefert immer einen Wert zwischen 0 und 3 und damit immer einen gültigen Farbcode für das "Basic"-Fenster.
Wenn man als Farbe den Code 4 angeben würde, so meldete der Basic-Interpreter einen Fehler, da hier nicht mehr Farben ansprechbar sind (was man aber ändern

könnte). Hinter dem Farbcode kann gemäß Syntaxdiagramm auch noch der Buchstabe "B" stehen, der durch ein Komma vom Farbcode getrennt wird. Das bedeutet, daß anstelle der Linie von x1,y1 nach x2,y2 ein Rechteck mit den entsprechenden Eckkoordinaten gezeichnet wird. Gibt man anstelle von "B" ein "BF" an, so wird das Rechteck ausgefüllt gezeichnet.
Nun sind wir bereit, eine mathematische Funktion einmal graphisch auszugeben. *Abb. 2.5.4* zeigt das Listing und im "Basic"-Fenster einen Ausschnitt der Funktion. Dabei wird eine komplexere Sinus-Funktion ausgegeben. Der Parameter dafür wird in der ersten Zeile nach der FOR-Anweisung in die Variable "wert" gelegt. In der Variablen "ergebnis" wird dann das Resultat gespeichert.
Nun muß man diesen Wert so umrechnen, daß er als Bildschirmkoordinate verstanden werden kann. Dazu ist die Funktion INT hilfreich, die wir bisher noch nicht verwendet haben. Sie macht aus einer Gleitkommazahl eine Ganze Zahl, indem die Nachkommastellen abgeschnitten werden. Da die Sinusfunktion und Cosinusfunktion immer einen Wert zwischen -1 und +1 liefert und somit auch die Multiplikation der beiden, kann man den Wertebereich sehr einfach auf Bildschirmgröße erhöhen. Hier wird die Amplitude durch Multiplikation mit 80 im Bereich -80 bis + 80 eingestellt und durch Addition des Wertes 100 liegt y im Bereich von 20..180, ist also somit im sichtbaren Bereich. Achtung, dadurch, daß die y-Koordinate mit dem Wert 0 am oberen Bildschirmrand liegt, wird hier die Funktion um die y-Achse gespiegelt dargestellt.

Aufgabe:
1. Fügen Sie eine Zeile ein, die die y-Achse getreu der Funktion darstellt.
Hinweis: y = .... y ....
2. Probieren Sie einmal andere Funktionen aus (EXP, LOG usw.).

Eine andere nützliche Aufgabe ist es, ein Gitter auf dem Bildschirm auszugeben. Wir wollen die Lösung dieser Aufgabe einmal schrittweise angehen.
Ein Gitter besteht aus horizontalen und vertikalen Linien. Also können wir die Aufgabe in zwei Teilaufgaben zerlegen:
1. Zeichnen der horizontalen Linien.
2. Zeichnen der vertikalen Linien.
Wie man horizontale Linien zeichnet, haben wir schon an dem Beispiel mit den farbigen Linien gesehen. Das ist also im Prinzip der erste Programmteil.
Für das Zeichnen von vertikalen Linien muß man nur noch x mit y vertauschen.
Jetzt muß man sich noch überlegen, wieviele Linien auf dem Bildschirm erscheinen sollen, welchen Abstand sie haben sollen und wo das Gitter anfangen soll. Wenn wir diese Angaben dem Benutzer überlassen wollen, so muß man die Werte mit einer INPUT-Anweisung einlesen. Dabei muß der Abstand für die x- und y-Richtung getrennt eingegeben werden können.
Das fertige Programm zeigt *Abb.2.5.5.* Geben Sie das Programm ein und starten es. Dann fragt der Computer zunächst nach der Anzahl der Linien. Geben Sie hier einmal den Wert 20 ein. Dann wird nach dem Abstand in x-Richtung gefragt. Hier geben Sie den Wert 4 ein. Für den Abstand in y-Richtung geben Sie schließlich 20 ein. Dann wird noch nach der Position der linken oberen Ecke in x-Richtung gefragt (x = ?) und Sie geben den Wert 100 ein. Für "y=?", also der linken oberen Ecke in y-Richtung geben Sie den Wert 20 ein. Danach wird das Gitter gezeichnet.

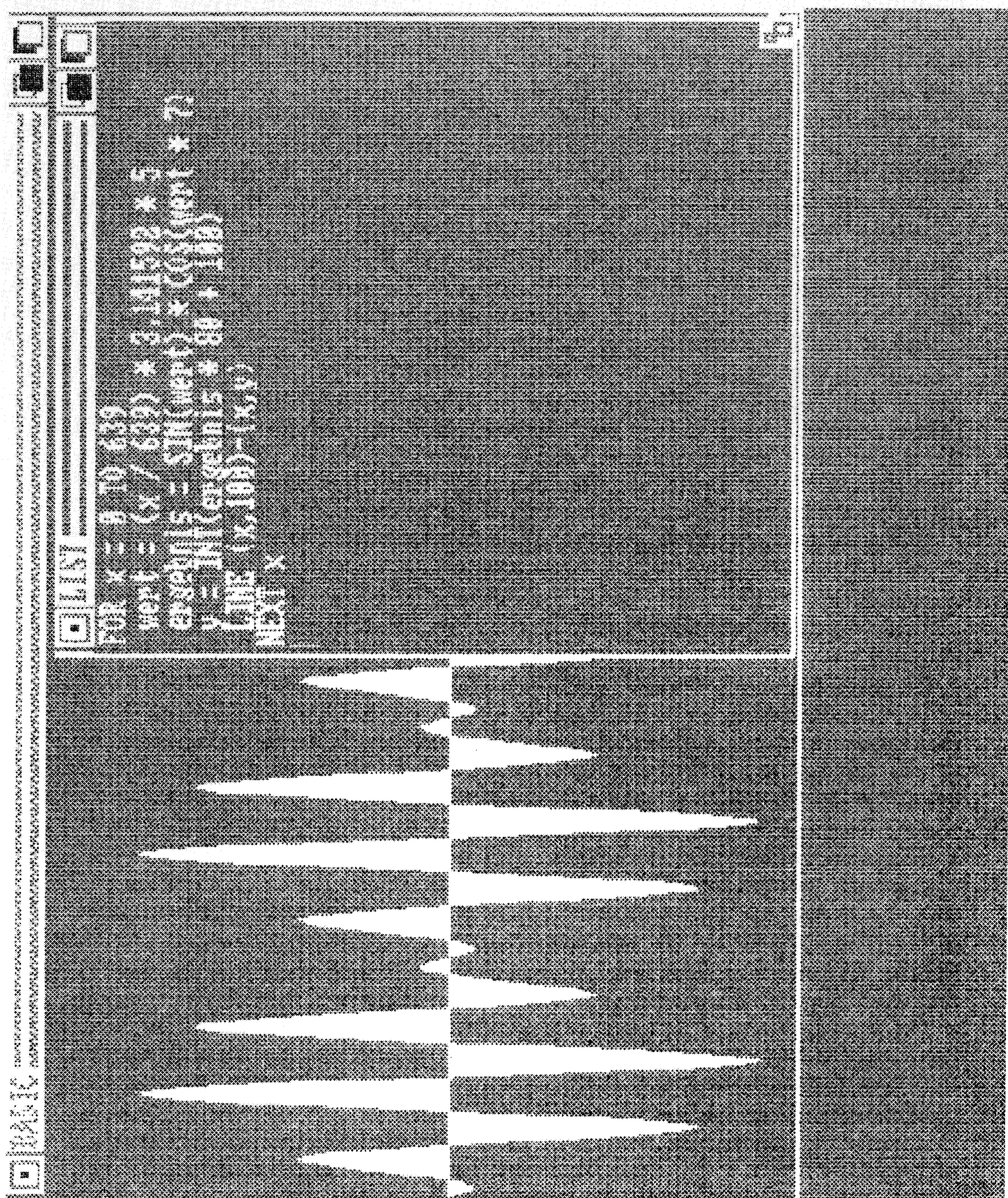



Abb. 2.5.4 Funktionen darstellen

```
INPUT "Anzahl Linien";anzahl
INPUT "Abstand x";xabstand
INPUT "Abstand y";yabstand
INPUT "x = ";xpos
INPUT "y = ";ypos
FOR y = ypos TO ypos+anzahl*yabstand STEP yabstand
 LINE (xpos,y)-(xpos+anzahl*xabstand,y)
NEXT y
FOR x = xpos TO xpos+anzahl*xabstand STEP xabstand
 LINE (x,ypos)-(x,ypos+anzahl*yabstand)
NEXT x
```

Abb. 2.5.5
Gitternetz

So können Sie einmal mit verschiedenen Werten experimentieren.
Es ergibt sich übrigens immer dann ein quadratisch aussehendes Gitter, wenn der Abstand in x-Richtung genau doppelt so groß wie der in y-Richtung ist, da der Bildschirm bei der Darstellung von 640 x 200 Punkten in x-Richtung ca. doppelt soviele Punkte aufweist.
FOR-Schleifen kann man auch ineinander verschachteln. Wenn Sie dazu das Syntaxdiagramm betrachten, steht dort zwischen FOR und NEXT ganz allgemein "Basic-Anweisungen", also auch weitere FOR-NEXT Schleifen. Dabei gibt es nur eine Einschränkung. Man darf nie die gleiche Schleifenvariable verwenden, sonst kann der Basic-Interpreter bei der NEXT-Anweisung ja nicht wissen, zu welcher FOR-Schleife sie gehört.
Eine entsprechende Fehlermeldung wird ggf. ausgegeben.
Beispiel:

```
FOR x=100 TO 200 STEP 4
 FOR y=50 TO 100 STEP 2
  LINE (x,y)-(x,y)
 NEXT y
NEXT x
```

Dieses Programm zeichnet ein Punktegitter auf dem Bildschirm. Dabei wird die innere Schleife 25 mal ausgeführt und die äußere führt die innerer Schleife auch 25 mal aus. Insgesamt wird der LINE-Befehl 25*25, also 625 mal ausgeführt. Hier ist der Startpunkt identisch mit dem Endpunkt, und der LINE-Befehl zeichnet dann einfach einen Punkt (es gibt auch noch andere Befehle, einen Punkt zu zeichnen).

## 2.6 Programmstrukturen - Verzweigung und Schleife

Neben der FOR-Schleife, die mit einer bekannten Anzahl von Schleifedurchläufen arbeitet, gibt es auch noch eine andere Schleifenart. Die WHILE-Schleife. Doch bevor wir uns daran machen, brauchen wir neue Operationen: Vergleiche und die Verknüpfungen.
Vergleiche:

| | |
|---|---|
| < | Kleiner als |
| = | Gleich |
| > | Größer als |
| <= | Kleiner Gleich |
| >= | Größer Gleich |

Probieren Sie die Vergleiche einmal aus:

**PRINT 1<2;1=1;1>2**

Als Ergebnis bekommen Sie:

**-1 -1 0**

Man kann übrigens auch Texte miteinander vergleichen dabei wird eine lexikalische Ordnung angenommen. Also "A"="A" ergibt -1, "Anton" < "Berta" ergibt auch -1, aber "A">"Ab" ergibt 0.

Der Wert -1 bedeutet "wahr" und der Wert 0 bedeutet "falsch". Warum gerade -1 als wahr verwendet wird, das hängt mit der Verwendung der Verknüpfungen zusammen.

Verknüpfungen:

| | |
|---|---|
| AND | Und-Verknüpfung. |
| OR | Oder-Verknüpfung |
| XOR | Exclusiv-Oder-Verknüpfung |
| EQV | Äquivalenz-Verknüpfung |
| IMP | Implikation |
| NOT | Nicht-Verknüpfung |

Die Nicht-Verknüpfung besitzt natürlich nur einen Operanden.

Lassen wir uns doch einmal per Programm eine Wahrheitstabelle ausgeben.

Dazu brauchen wir zwei Variablen, mit denen alle Kombinationen gebildet werden. Mit zwei verschachtelten FOR-Schleifen kann man dann alle Kombinationen durchwählen. Dabei ist -1 wahr und 0 falsch als Eingangskombination.

Wahrheitstafelprogramm:

```
FOR op2=0 TO -1 STEP -1
  FOR op1=0 TO -1 STEP -1
    PRINT op2;op1;" : ";op1 AND op2; op1 OR op2;
    PRINT op1 XOR op2; op1 EQV op2; op1 IMP op2;
    PRINT NOT op1
  NEXT op1
NEXT op2
```

Hier wurden drei PRINT-Anweisungen verwendet um eine Ausgabezeile aufzubereiten, man kann jedoch alle Operationen auch hinter eine PRINT-Anweisung schreiben.

Das Ergebnis auf dem Bildschirm:

```
 0  0 :  0  0  0 -1 -1 -1
 0 -1 :  0 -1 -1  0  0  0
-1  0 :  0 -1 -1  0 -1 -1
-1 -1 : -1 -1  0 -1 -1  0
```

Aufgaben:

1. Erweitern Sie das Programm so, daß als Ergebnis 1 für wahr steht. Hinweis: verwenden Sie dazu die ABS-Funktion.
2. Erweitern Sie das Programm um Überschriften, so daß man erkennen kann, welche Verknüpfungen den Spalten zugeordnet sind.
3. Experimentieren Sie einmal mit anderen Eingangswerten bei den Verknüpfungen, Beispiel: was ergibt 5 AND 6 und warum?

Nun aber zurück zur WHILE-Schleife. *Abb.2.6.1* zeigt das dazugehörige Syntaxdiagramm. Die Struktur ähnelt der FOR-Schleife, jedoch gibt es hier keine

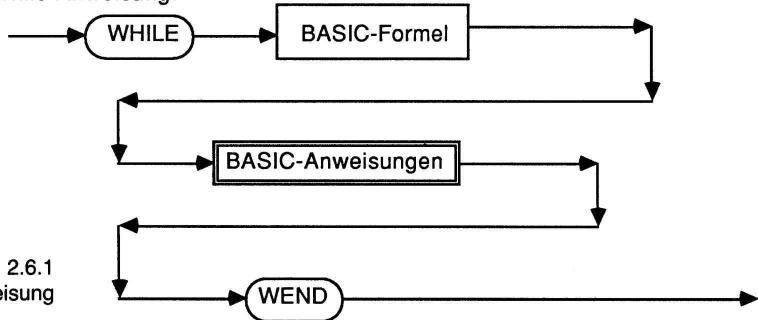



Abb. 2.6.1
Die WHILE-Anweisung

Schleifenvariable. Dafür entscheidet die Angabe hinter dem WHILE-Befehl über die Schleifenausführung. Es gilt dabei: Solange die Basic-Formel den Wert WAHR besitzt, wird die Schleife, also die darin liegenden Basic-Anweisungen, ausgeführt. Beispiel:

```
i = 2
WHILE  i<=2^20
  PRINT i
  i = i * 2
WEND
```

Man erhält eine Tabelle mit Zweierpotenzen. Die Anweisung i = i * 2 bewirkt, daß der alte Wert von i mit 2 multipliziert wird und danach erneut der Variablen i zugewiesen wird.
Die WHILE-Schleife kann man natürlich auch verschachteln, so wie wir es von der FOR-Schleife schon kennen.
Ist die Bedingung schon beim ersten Mal nicht erfüllt, so werden die Anweisungen in der WHILE-Schleife gar nicht erst durchlaufen.
Nun will man nicht nur Schleifen bauen, sondern auch Verzweigungen realisieren. Dafür gibt es eine gesonderte Anweisung, die IF-Anweisung. *Abb.2.6.2* zeigt das Syntaxdiagramm. Bitte nicht gleich verzweifeln, die IF-Anweisung bietet ungleich mehr Möglichkeiten, als die Anweisungen, die wir bisher kennengelernt haben. Am leichtesten lernt man durch Beispiele und daher fangen wir mit einem einfachen Beispiel an:

```
INPUT a
IF a=1 THEN
 PRINT "A ist EINS"
ELSE
 PRINT "A ist nicht EINS"
END IF
```

Starten Sie das Programm. Es erscheint ein Fragezeichen. Sie können dann einen Wert eingeben. Wenn Sie die Zahl 1 eintippen, so wird anschließend der Text "A ist EINS" ausgegeben, wenn Sie eine andere Zahl eintippen, so wird der Text "A ist nicht EINS" ausgegeben. Das Programm realisiert eine Verzweigung. Die Befehle hinter dem THEN und vor dem ELSE werden nur dann ausgeführt, wenn die

IF-Anweisung:

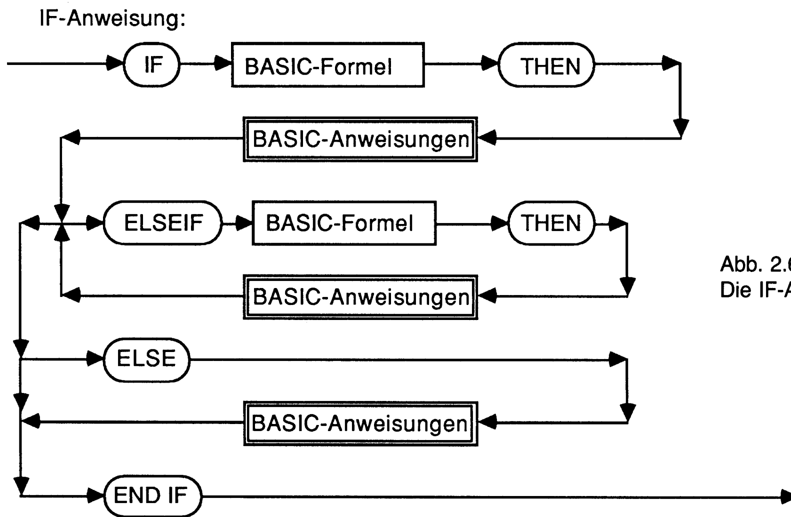



Abb. 2.6.2
Die IF-Anweisung

Bedingung bei der IF-Anweisung einen Wert "wahr" besitzt. Die Befehle hinter der ELSE-Anweisung und vor der END IF - Anweisung werden nur dann ausgeführt, wenn die Bedingung bei der IF-Anweisung einen Wert "falsch" besitzt.
Übrigens muß man den ELSE-Teil nicht unbedingt hinschreiben (siehe Syntaxdiagramm), wenn man ihn nicht braucht. Eine weitere Möglichkeit ist es, im THEN-Teil ein oder mehrere ELSEIF-Teile dazuzuschreiben. Man kann damit Mehrfachverzweigungen realisieren.
Beispiel:

```
INPUT a
IF a=1 THEN
 PRINT "EINS"
ELSEIF a=2 THEN
 PRINT "ZWEI"
ELSEIF a=3 THEN
 PRINT "DREI"
ELSE
 PRINT "Nicht 1 bis 3"
END IF
```

Achtung: ELSEIF schreibt man zusammen, obwohl es im deutschen Amiga-Basic-Handbuch getrennt geschrieben wurde. Hingegen END IF sind zwei Worte und man muß ein Leerzeichen dazwischen schreiben. Wenn man das nicht tut, erhält man einen Syntax-Fehler.
Mit den bisher vorgestellten Befehlen kann man schon recht anspruchsvolle Programme realisieren.
So wollen wir einfach mal eine kleine Anwendung programmieren.
Wir wollen ein kleines Abrechnungsprogramm schreiben. Man soll dabei die Stückzahl einer Ware und den Preis eingeben. Am Schluß soll der Rechner die Summe ausgeben.

Wie geht man dabei vor? Zunächst sollte man versuchen, eine Struktur zu finden. Hier kann man erkennen, daß die Eingabe der Anzahl und des Preises ein wiederholter Vorgang ist. Also braucht man eine Schleifenstruktur. Welche Schleife soll man verwenden? Das hängt davon ab, wann die Schleife abgebrochen werden soll. Eine feste Anzahl von Durchläufen kann man hier nicht erwarten, denn die Anzahl der Waren kann unterschiedlich sein, also sollte man nicht die FOR-Schleife verwenden. Auch wäre es mühsam, vorher immer die Anzahl der unterschiedlichen Waren einzugeben. Bleibt die WHILE-Schleife. Wir müssen dazu ein Abbruchkriterium finden. Man kann die Schleife zum Beispiel abbrechen, wenn man die Zahl 0 als Eingabe für den Preis oder als Anzahl eingibt. Nun muß man bei der WHILE-Schleife die Bedingung aber am Anfang der Schleife abfragen. Daher muß eine Eingabe schon vor Beginn der Schleife gemacht werden. Ferner braucht man für das Programm eine Variable für die Summe und je eine für die Anzahl und den Preis.
Das Programm sieht dann so aus:

```
summe = 0
INPUT "Anzahl:";anzahl
INPUT "Preis:";preis
WHILE (anzahl <> 0) AND (preis <> 0)
  summe = summe + anzahl * preis
  INPUT "Anzahl:";anzahl
  INPUT "Preis:";preis
WEND
PRINT "Total:";summe;"DM"
```

Wenn man das Programm startet, so fragt es nach der Anzahl. Dann gibt man die Anzahl der Waren ein. Anschließend fragt es nach dem Preis, den man dann auch eingibt. Dann wird wieder die Anzahl erfragt usw. Wenn man als Anzahl 0 oder als Preis 0 eingibt, so wird das Programm beendet, sobald ein erneuter Schleifendurchlauf erfolgt, und die Summe ausgegeben.
Wenn man als Anzahl 0 eingibt, wird daher noch nach dem Preis gefragt. Wir wollen das ändern und das Programm verbessern.
Jetzt brauchen wir eine IF-Anweisung. Es darf die Frage nach dem Preis nur dann erfolgen, wenn eine Anzahl ungleich Null eingegeben wurde, also IF anzahl ungleich Null, dann INPUT preis.
Das Programm sieht dann vollständig so aus:

```
summe = 0
INPUT "Anzahl:";anzahl
INPUT "Preis:";preis
WHILE (anzahl <> 0) AND (preis <> 0)
  summe = summe + anzahl * preis
  INPUT "Anzahl:";anzahl
  IF anzahl <> 0 THEN
     INPUT "Preis:";preis
  ENDIF
WEND
PRINT "Total:";summe;"DM"
```

Einen ELSE-Teil brauchen wir hier nicht. Wenn man jetzt 0 als Anzahl eingibt, so wird die Schleife ohne erneute Abfrage des Preises beendet. Dabei geschieht das nur, wenn man mindestens einmal Anzahl und Preis korrekt eingegeben hat.
Aufgaben:

1. Was passiert, wenn man einen negativen Preis eingibt, also z.B. -3.10. Wozu kann man diesen Effekt ausnutzen?

2. Erweitern Sie das Programm, so daß man bei Eingabe einer negativen Anzahl durch eine Fehlermeldung gewarnt wird und man danach die Eingabe erneut durchführen kann.

3. Geben Sie eine Fehlermeldung aus, wenn die Gesamtsumme kleiner oder gleich 0 ist.

## 2.7 Datenfelder

Wenn man Werte im Speicher festhalten wollte, so mußte man bisher immer einen jeweils neuen Variablennamen wählen. Das wird jetzt anders.
Mit dem Befehl **DIM** kann man Speicherplatz für Datenfelder reservieren.
Dazu gibt man hinter dem Namen DIM eine Liste von Variablennamen, die in Klammern die Feldgröße enthalten, an.
Beispiel:

**DIM preise(100)**

Dadurch wird ein Feld mit 101 Elementen reserviert. Man kann einzelne Speicherzellen später dadurch anwählen, daß man in Klammern den Index angibt:

**preise(10)**

Wählt ein bestimmtes Element aus, das man dann wie zuvor mit den einfachen Variablen in Rechnungen verwenden kann.
Es sind 101 Elemente, da man mit dem Index bei 0 zu zählen beginnt.
Das erste Element ist also **preise(0)** und das letzte Element ist **preise(100).**
Mit Textvariablen geht das entsprechend, nur daß das Dollarzeichen hinter dem Namen steht.
Als Aufgabe wollen wir ein Programm schreiben, das eine Liste von Waren mit den dazugehörigen Preisen einliest. Anschließend soll man durch Eingabe des Warennamens den Preis erfahren.
Das Programm besteht aus zwei Teilen. Im ersten Teil werden alle Namen der Waren und die dazugehörigen Preise eingelesen. Im zweiten Teil wird in einer Schleife immer nach dem Warennamen gefragt und dann, falls die Ware gefunden wurde, der Preis ausgegeben.
Das fertige Programm zeigt *Abb. 2.7.1*.
Es werden zwei Felder angelegt. Das erste Feld mit dem Namen "waren$()" speichert die Namen der einzelnen Waren, das zweite Feld mit dem Namen "preise()" speichert die dazugehörigen Preise.
Beide Felder wurden hier mit der Dimension 200 angelegt, das reicht dann für 200 Wareneinträge. Will man mehr, so muß man den Wert entsprechend erhöhen.
Die nächste Zeile im Programm beginnt mit einem neuen Befehl REM. Das ist die Abkürzung für REMARK und heißt zu deutsch Bemerkung. Alles, was hinter REM

```
DIM waren$(200),preise(200)
REM Warenliste einlesen
i = 0
INPUT "Ware:",waren$(i)
INPUT "Preis:",preise(i)
WHILE (waren$(i) <> "") AND (i < 200)
 i = i + 1
 INPUT "Ware:",waren$(i)
 IF waren$(i) <> "" THEN
  INPUT "Preis:",preise(i)
 END IF
WEND
REM nun Abfrage der Warenliste
nameware$ = "nix"
WHILE nameware$ <> ""
 INPUT "Welche Ware:",nameware$
 i = 0
 WHILE (waren$(i) <> nameware$) AND (i < 200)
   i = i + 1
 WEND
 IF i = 200 THEN
  PRINT "Ware nicht im Katalog."
 ELSE
  PRINT "Ware ";nameware$;" kostet ";preise(i);"DM"
 END IF
WEND
```

Abb. 2.7.1
Warenkalkulation

steht wird vom Basic-Interpreter ignoriert. Verwendet wird dieser Befehl um Kommentare ins Programm zu schreiben, so daß man es besser verstehen kann.
Nach dem REM-Befehl wird die Variable i mit dem Wert 0 vorbelegt. Die Variable i wird als Indexzähler verwendet. Nun werden die beiden ersten Wertepaare eingelesen, also Name der Ware und Preis. Dann folgt der Eintritt in die Schleife. Die Schleife wird solange ausgeführt, wie man einen Namen eintippt. Drückt man nur die Returntaste, so wird die Schleife beendet.
In der Schleife wird zunächst der Index um eins erhöht und dann erneut Ware und Preis eingelesen, falls man nicht nur Return eingegeben hat.
Nach Beendigung der Schleife gelangt man zur Abfrage. Dabei wird eine weitere Variable mit dem Namen "nameware$" zunächst mit dem Text "nix" belegt. Dies ist eine andere Möglichkeit, den vorzeitigen Schleifenabbruch zu Verhindern. Dann braucht man nämlich nur einen INPUT in der Schleife. Die Schleife wird auch diesmal solange ausgeführt, bis man nur die RETURN-Taste als Antwort auf die Eingabe-Anforderung drückt.
Hat man eine Ware eingegeben, so beginnt die Suche. Zunächst wird wieder eine Indexvariable mit dem Wert 0 belegt, die Suche soll beim ersten Element beginnen. Solange der Variableninhalt von "nameware$" ungleich dem Inhalt des angewählten Feldelementes "waren$(i)" ist, wird gesucht. Die Suche wird aufgegeben, wenn man alle 200 Elemente des Feldes durchsucht hat. Übrigens sind am Anfang alle Feldelemente mit einem leeren Text ("") bzw. mit 0 bei Zahlvariablen vorbelegt.
Ist der Wert i nach Ende der Schleife auf dem Wert 200 angekommen, so wurde die

Ware nicht gefunden und ein entsprechender Text wird ausgegeben. Wurde sie aber gefunden, so ist die Schleife vorher abgebrochen worden und der Preis kann ausgegeben werden.
Aufgaben:
1. Gestalten Sie das Programm so um, daß maximal nur soviele Elemente verglichen werden, wie zuvor eingegeben worden sind.
2. Überlegen Sie, wie man das Suchen optimaler gestalten kann.
Eine andere interessante Anwendung ist die Auswertung von Zufallszahlen. Sie spielen bei der Programmierung z.B. von Spielen oder Statistiken häufig eine wichtige Rolle. Mit der Funktion **RND()** kann man eine Zufallszahl zwischen 0 und 1 bekommen. Als Argument wird eine Zahl mitgegeben. Ist sie positiv, so erhält man die nächste Zufallszahl. Wenn die Zahl=0 ist, erhält man den letzten Wert nochmals, bei einer negativen Zahl wird der Anfangswert einer Zufallssequenz neu gesetzt.
Wir wollen ein Programm schreiben, mit dem man die Verteilung der Zufallszahlen betrachten kann. Die Zufallszahlen sind nämlich in Wirklichkeit keine echten Zufallswerte, sondern werden berechnet. Der Zufallszahlen-Generator sollte dabei eine möglichst gleichmäßige Verteilung über den Zahlenbereich liefern. RND() liefert eine Gleitkommazahl im Bereich 0 bis 1, dabei können natürlich fast beliebig viele Zahlen auftauchen, daher werden wir den Bereich für die Untersuchung auf einen kleinen Bereich beschränken. Das geschieht durch Multiplikation mit einer Zahl und anschließendem Abschneiden der Nachkommastellen. *Abb. 2.7.2* zeigt das komplette Listing. Die Zufallszahl wird mit dem Wert 100 multipliziert und das Ergebnis als Index in ein Feld verwendet.
Der Inhalt wird immer um Eins erhöht, wenn der Index des Feldelementes auftaucht. Zusätzlich wird der aktuelle Zählerstand graphisch auf dem Bildschirm ausgegeben. Dazu dient der Linien-Befehl.
Das Ganze wird in einer Schleife, hier 5000 mal, wiederholt. Je größer die Anzahl der Schleifendurchläufe, desto genauer das Ergebnis. Die Linien werden dann allerdings auch höher.
Aufgabe:
Verwenden Sie einmal die Formel r = INT(RND(1) * 50 + RND(1)* 50).
Was sagt das Ergebnis?

Datenlisten kann man übrigens in Basic auch direkt im Programm unterbringen. Dazu gibt es zwei Befehle:
**DATA** und **READ.** DATA erlaubt es, Werte als Liste im Programm abzulegen und mit READ kann man sie dann nacheinander einlesen.

```
REM Zufallsverteilung
DIM werte(100)
FOR i= 1 TO 5000
 r = INT(RND(1) * 100)
 werte(r) = werte(r) + 1
 x = (r * 3) + 20
 LINE (x,150)-(x,150-werte(r))
NEXT i
```

Abb. 2.7.2
Zufallsverteilung

Beispiel:

```
FOR i=1 TO 4
 READ a$,a
 PRINT a$,a
NEXT i
DATA "Hallo",5,"wie",10,"geht",23.34,"es",-3
```

Die Datenwerte werden nacheinander eingelesen und auf dem Bildschirm ausgedruckt. Dabei muß man darauf achten, daß nicht mehr Daten eingelesen werden, als vorhanden sind, sonst erscheint die Fehlermeldung "Out of DATA". Auch die Abfolge der Daten ist wichtig. Da hier zuerst eine Textvariable gelesen wird und dann eine Zahlvariable, muß in der DATA-Anweisung auch zuerst ein Text und dann eine Zahl folgen. Wenn man mehr Daten hat, als in eine Zeile passen, dann kann man auch weitere DATA-Anweisungen folgen lassen. Die READ-Anweisung holt sich die Daten nacheinander beginnend bei der ersten gefundenen DATA-Anweisung bis keine DATA-Anweisung mehr gefunden ist. Mit dem Befehl RESTORE kann man den Lesezeiger auch wieder zurücksetzen (siehe Basic-Handbuch).

## 2.8 Unterprogramm-Technik

Wenn man Programmteile häufig braucht, so ist es sehr mühsam sie jedesmal wieder neu einzutippen und das Programm wird dadurch immer größer und größer. Dazu gibt es einen praktischen Mechanismus, das Unterprogramm. Man kann ein Unterprogramm schreiben und es vom Hauptprogramm aus aufrufen. Ein Unterprogramm sieht aus wie ein normaler Programmteil, nur daß als letzter Befehl **RETURN** steht. Das bedeutet soviel wie Rückkehr. Man ruft ein Unterprogramm mit dem Befehl **GOSUB** auf. Dazu benötigt GOSUB noch einen Parameter, der sagt, wo das Unterprogramm beginnt. Beim Amiga geht das ganz elegant. Dazu wird am Beginn des Unterprogramms ein Name gefolgt von einem Doppelpunkt geschrieben. Der Aufruf erfolgt dann durch Angabe des Namens. Als Beispiel soll eine Eingabe mehrfach verwendet werden, die abfragt, ob der eingegebene Wert negativ ist und eine Fehlermeldung dabei ausgibt. Das Hauptprogramm soll zwei Werte anfordern.

Programm:

```
REM Hauptprogramm
GOSUB wertholen
a = wert
GOSUB wertholen
b = wert
PRINT a,b
END
REM hier beginnt das Unterprogramm
wertholen:
wert = -1
WHILE wert < 0
 INPUT "Wert:";wert
  IF wert < 0 THEN
```

```
    PRINT "Wert kleiner 0, nochmals eingeben bitte!"
  END IF
WEND
RETURN
```

Im Unterprogramm wird die Variable wert zunächst mit -1 belegt, so daß die While-Schleife mindestens einmal durchlaufen wird. Dort wird dann der Wert eingelesen. Wurde eine negative Zahl eingegeben, so wird die Schleife erneut durchlaufen, zuvor erfolgt noch eine Meldung an den Benutzer.
Der GOSUB-Befehl kehrt nach der Abarbeitung des Unterprogramms wieder an die alte Programmstelle zurück. Das Ergebnis der Eingabe ist hier in der Variable "wert" gespeichert und kann dann im Hauptprogramm beliebig verwendet werden.
Am Schluß des Hauptprogramms steht noch der Befehl END, der das Programmende angibt. Läßt man ihn weg, so würde anschließend das Unterprogramm ausgeführt und dann bekommt man eine Fehlermeldung, sobald der Befehl RETURN an die Reihe kommt, denn wohin soll das Unterprogramm zurückkehren, wenn kein GOSUB gegeben wurde?

Bei dieser Art Unterprogramme gibt es noch ein kleines Problem. Wenn man Variable im Unterprogramm, z.B. zur Speicherung von Zwischenergebnissen verwendet, so sollten sie im Hauptprogramm nicht vorkommen. Verwendet man solch eine Variable, so wird der alte Inhalt überschrieben. Wenn diese Variable im Hauptprogramm verwendet wurde, so ist nach Rückkehr aus dem Unterprogramm der alte Wert verschwunden. Dies kann zu Programmfehlern führen, die nur schwer zu finden sind.
Beim Amiga-Basic gibt es noch eine andere Art Unterprogramme zu formulieren, die diesen Nachteil vermeidet. Dazu wird der Anfang des Unterprogramms mit dem Befehl **SUB name (parameter) STATIC** eingeleitet und das Ende durch **END SUB**. Dabei ist "name" der Name des Unterprogramms. Auf die Bedeutung von "parameter" kommen wir später noch.
Der Aufruf des Unterprogramms erfolgt durch **CALL name (parameter)** oder sogar ohne CALL durch **name parameter**. Dabei bleiben die "parameter" ohne Klammer. Wenn keine Parameter vorhanden sind, kann man sie immer inclusive Klammern weglassen.
Beispiel:

```
rem Hauptprogramm
meinunterprogramm
meinunterprogramm
SUB meinunterprogramm STATIC
  PRINT "Aufruf erfolgt."
END SUB
```

Nach dem Start des Programms wird der Text "Aufruf erfolgt." zweimal ausgegeben. Übrigens kann man sich hier den Befehl END sparen, da das Unterprogramm nicht aus Versehen durchlaufen werden kann, sondern nur durch Angabe des Namens. Alle Variablen, die man nun innerhalb des Unterprogramms verwendet gehören ausschließlich diesem Unterprogramm. Dadurch werden Namenskonflikte vermieden.

Beispiel:

```
a = 1
aufruf
PRINT "Hauptprogramm:", a
SUB aufruf STATIC
  a = 3.123
  PRINT "Unterprogramm:",a
END SUB
```

Wenn man das Programm startet, so erhält man die Ausgaben:

```
Unterprogramm   3.123
Hauptprogramm   1
```

Der Wert der Variablen "a", die im Hauptprogramm verwendet wurde, blieb erhalten.
Nun kommt hier gleich eine Frage auf. Wie kann man Werte vom Unterprogramm in das Hauptprogramm bekommen, wenn alle Variablen des Hauptprogramms unerreichbar für das Unterprogramm sind.
Dazu gibt es zwei Möglichkeiten. Die erste ist die Verwendung von Parametern.

Beispiel:

```
REM Hauptprogramm
einlesen a
einlesen b
ausgeben a+b
REM Unterprogramme folgen hier
SUB einlesen(wert) STATIC
  INPUT "Zahl eingeben:";wert
END SUB
SUB ausgeben(wert) STATIC
  PRINT "Ergebnis:";wert
END SUB
```

Wenn man mehr als einen Parameter verwenden will, so werden die einzelnen Parameter durch Kommas getrennt. Achtung, wenn man das Unterprogramm durch CALL aufruft, so muß man alle Parameter in Klammern angeben.
Wenn man verhindern will, daß ein Parameter im Unterprogramm verändert wird, so kann man ihn zusätzlich einklammern.
einlesen (a) würde a unverändert lassen, genauso wie CALL einlesen ((a)).
Dabei sind zwei Klammern nötig, die erste braucht man immer bei der Verwendung von CALL und die zweite verhindert das Verändern der Variablen. Dies ist allerdings bei unserem Programm "einlesen" nicht sinnvoll, da man ja einen Wert erhalten will.
Eine andere Möglichkeit ist die Verwendung des Befehls SHARED. Hinter SHARED schreibt man alle Variablen, auf die man vom Unterprogramm aus zugreifen möchte. Dabei muß der Befehl SHARED der erste Befehl nach SUB ... STATIC sein.

Beispiel:

```
a = 1
aufruf
PRINT a
SUB aufruf STATIC
 SHARED a
 PRINT a
 a  = 5
END SUB
```

Die Verwendung von Unterprogrammen dieser Art empfiehlt sich auch um Programme lesbarer zu machen. Ferner kann man mit ihnen quasi eigene Befehle bauen und sich so einen eigenen Wortschatz konstruieren.

## 2.9 Dateien - Speichern und Laden von Daten und Programmen

Wenn man den Rechner ausschaltet, sind alle Programme und Daten verschwunden, die sich zuvor im Hauptspeicher des Rechners befunden haben.
Will man Programme oder Daten dauerhaft anlegen, so muß man sie auf einem Datenträger speichern. Das kann eine Floppy sein oder gar eine Harddisk, die besonders viel Speicherplatz hat. Auf einer Diskette kann man beim Amiga bis zu 880 KByte speichern, bei der Harddisk hängt das vom Laufwerkstyp ab, typisch sind aber 20 MByte bis 50 MByte.
Der Vorteil des Floppy-Laufwerks liegt darin, daß man die Disketten, also den sogenannten Datenträger auswechseln kann. Bei der Harddisk geht das nicht, der Datenträger ist nicht auswechselbar.
Zunächst einmal widmen wir uns dem Speichern von Programmen. Vielleicht haben Sie schon im Handbuch nachgesehen, es ist der Befehl **SAVE**. Save heißt soviel wie sichern. Damit man Programme später wiederfindet, muß man sie mit einem Namen versehen. Daher verlangt SAVE noch einen Text als Parameter. Geben Sie einmal ein kleines Programm ein. Dann klicken Sie das Basic-Fenster an und tippen SAVE "mein Programm". Dabei muß man darauf achten, wohin das Programm gespeichert werden soll. Wenn Sie nur ein Amiga-Laufwerk haben, wird das Programm ggf. auf die Amiga-Basic-Extras-Diskette gespeichert. Dies geht aber nur, wenn dort kein Schreibschutz gesetzt ist. Der Schreibschutz ist eine Art Schalter an der Floppy, mit dem ein kleines Loch am Rand der Diskette freigegeben wird. Ist das Loch offen, so ist der Schreibschutz gesetzt und man kann keine Daten darauf ablegen. Das Betriebssystem des Amiga erkennt dies und meldet in einem eckigen Kasten links oben "Volume .... is write protected". Nun können Sie entweder die Diskette herausnehmen und den Schreibschutz entfernen und wieder einlegen  oder "Cancel" anklicken, wenn Sie z.B. gar nicht mehr speichern wollen. Danach erhalten Sie noch eine Fehlermeldung vom Basic.
Um herauszufinden, welche Dateien schon auf der Diskette sind, kann man den Befehl **FILES** eintippen. Es wird dann ein Inhaltsverzeichnis auf dem Bildschirm ausgegeben. Die gespeicherte Datei "mein Programm" taucht zweimal auf. Einmal so wie Sie sie geschrieben haben und dann noch als "mein Programm.info". Dies ist eine Datei, die der Amiga braucht um ein ICON auf dem Bildschirm auszugeben,

wenn Sie das Programm z.B. später einmal direkt durch anklicken starten wollen.
Das Programm können Sie wieder laden, indem Sie den Befehl
**LOAD "mein Programm"** eintippen. Achtung! Dabei müssen Sie den Namen genauso schreiben, wie Sie es beim SAVE-Befehl getan haben, sonst wird das Programm nicht gefunden.
Der Amiga hat ein sehr komfortables Dateiverwaltungs-System, in dem man sogenannte Unterverzeichnisse, auch Schubladen genannt, anlegen kann.
Um von einer Schublade zur nächsten zu wechseln, gibt es einen Befehl:
**CHDIR "pfadname"**. Beispiel:
Mit CHDIR "df0:" können Sie das interne Laufwerk ansprechen, wobei Sie sich im Pfad an äußerster Stelle befinden. Geben Sie nun den Befehl FILES ein, so erhalten Sie eine Liste aller Dateien. Alle vorhandenen Unterverzeichnisse werden dabei auch ausgegeben und sind in eckigen Klammern gesetzt. Nun können Sie ein solches Verzeichnis anwählen, indem Sie mit CHDIR "name" den Namen dieses Verzeichnisses angeben. Dort können wieder weitere Verzeichnisse angelegt sein, die man wieder angeben kann.
Die Unterverzeichnisse kann man natürlich auch selbst anlegen.
Wenn man einen Pfad angeben will, ohne CHDIR zu verwenden, so kann man das tun, indem man die Namen durch das Zeichen "/" trennt. Beispiel:
**LOAD "df1:unterverzeichnis/mein Programm"**
Hier wird noch mit "df1:" zusätzlich das Laufwerk 1 angewählt, das natürlich dann auch vorhanden sein muß.
Will man den Namen eines gespeicherten Programms nachträglich ändern, so verwendet man den Befehl: **NAME "alter name" AS "neuer name"**. Will man eine Datei loswerden, so kann man sie mit **KILL "name"** löschen.
Speichern und Laden von Programmen können Sie auch mit Hilfe der Menüleiste, dort findet man sie unter "Project", wobei das Laden mit "open" geschieht. Der Punkt "save as" erlaubt die Angabe eines Namens, mit "save" wird eine Datei abgespeichert, wenn der Name schon bekannt ist.
SAVE besitzt noch eine zweite Form: **SAVE "name",A.**
Das "A" steht für ASCII. Das Programm wird als Textdatei gespeichert. Man kann es dann später durch andere Programme weiterverarbeiten. Normalerweise werden Basic-Programme in einer internen Form abgespeichert. Dabei sind Befehle wie PRINT nicht als Text mit z.B. 5 Buchstaben abgelegt, sondern durch einen besonderen Code, der nur ein Zeichen belegt.
Wir wollen uns aber jetzt der Dateiverwaltung von Daten widmen. Dazu brauchen wir ein paar neue Befehle, die Sie anhand eines Beispiel kennenlernen sollen.
*Abb.2.9.1* zeigt das Programmlisting. Das Hauptprogramm besteht aus eine Schleife, in der zunächst einmal mit drei Print-Anweisungen ein kleines Menü

```
REM Telefonbuch - Verwaltung 1
WHILE was <> 3
 PRINT "1 = anlegen"
 PRINT "2 = abfragen"
 PRINT "3 = Programmende"
 INPUT was
 IF was = 1 THEN
  anlegen
```

```
 ELSEIF was = 2 THEN
  abfragen
 END IF
WEND

SUB anlegen STATIC
 schreiben
 n$ = "nix"
 WHILE n$<>"END"
  INPUT "Name:";n$
  INPUT "Telefonnumer";t$
  eintragen n$,t$
  PRINT "----------------"
 WEND
 schliessen
END SUB

SUB abfragen STATIC
 nam$ = "nix"
 WHILE nam$<>"END"
  INPUT "Name suchen:",nam$
  lesen
  INPUT #1,n$,t$
  WHILE (n$ <> nam$) AND (EOF(1) = 0)
   INPUT #1,n$,t$
  WEND
  IF (EOF(1) <> 0) THEN
   PRINT "Name nicht vorhanden"
  ELSE
   PRINT "Telefonnr:";t$
  END IF
  schliessen
 WEND
END SUB


SUB schreiben STATIC
 OPEN "telefon" FOR APPEND AS 1
END SUB

SUB lesen STATIC
 OPEN "telefon" FOR INPUT AS 1
END SUB

SUB schliessen STATIC
 CLOSE 1
END SUB


SUB eintragen(nam$,tel$) STATIC
 WRITE#1,nam$
 WRITE#1,tel$
END SUB
```

Abb. 2.9.1
Telefonbuch-Verwaltung

ausgegeben wird. Mit Funktion 1 soll man das Telefonbuch erweitern können, Funktion 2 soll eine Abfrage ermöglichen und mit 3 kann man das Programm beenden. Wenn man die Zahl 1 eingegeben hat, so wird das Unterprogramm "anlegen" aufgerufen. Hat man die Zahl 2 eingegeben, so wird das Unterprogramm "abfragen" aufgerufen.
Nun zu den Unterprogrammen.
Im Unterprogramm "anlegen" wird zunächst ein weiteres Unterprogramm mit dem Namen "schreiben" aufgerufen. Es soll die Ausgabe auf die Datei vorbereiten. Danach beginnt eine Schleife. Sie wird solange durchlaufen, bis man den Namen "END" eingegeben hat. In der Schleife wird der Name des Telefoninhabers abgefragt und anschließend seine Telefonnummer, die hier auch in eine Textvariable abgelegt wird, so daß man auch Klammern usw. eingeben kann. Dann wird das Unterprogramm "eintragen" mit zwei Parametern, nämlich dem Namen, der in n$ steht und der Telefonnummer, die in t$ steht, aufgerufen.
Damit man auf dem Bildschirm eine bessere Trennung hat, wird dann noch eine Print-Anweisung ausgeführt und es werden ein paar Minuszeichen ausgegeben. Wenn die Schleife durch Eingabe von "END" beendet wurde, so wird noch das Unterprogramm "schliessen" aufgerufen.
Nun zu den Unterprogrammen, die die eigentlichen Dateibefehle beinhalten.
Im Unterprogramm "schreiben" wird die Datei eröffnet. Dazu gibt es den Befehl **OPEN**. Er erhält als Parameter zunächst den Namen der Datei und dann die Sequenz **FOR APPEND AS 1**. Dies bedeutet: die Datei wird so geöffnet, daß man Daten immer anhängen kann, unser Telefonbuch soll ja erweiterbar sein, und daß sie mit der Kennung 1 ansprechbar ist. Die Kennung braucht man, da man mehr als eine Datei ansprechen kann, die Zahl kann zwischen 1 und 255 gewählt werden. Außer APPEND sind auch OUTPUT, wenn eine Datei vor dem Schreiben automatisch gelöscht werden soll, oder INPUT für das Lesen zulässig.
Im Unterprogramm "lesen" z.B. wird unsere Datei zum Lesen angemeldet. Das Unterprogramm brauchen wir aber erst für "abfragen".
Nun ist noch "schliessen" wichtig. Mit dem Befehl "CLOSE 1" wird die Datei geschlossen. Das bedeutet: eventuell im Speicher stehende Daten werden endgültig auf die Diskette übertragen und das Inhaltsverzeichnis der Diskette wird auf den neuen Stand gebracht. Danach kann man die Datei nicht mehr unter der Kennung 1 erreichen.
Das Unterprogramm "eintragen" schließlich schreibt die Daten auf die Datei. Dabei legt der Befehl **WRITE#1**, ... die Daten ab.Man kann übrigens auch **PRINT#1**, ... schreiben, was jedoch andere Effekte hat.
WRITE legt alle Daten so ab, daß man sie nachher mit einem INPUT wieder genauso lesen kann. Wenn man PRINT verwendet, so werden z.B. Kommas nicht ausgegeben oder Texte ohne Anführungszeichen ablegt. Beim Einlesen durch INPUT ist das dann nicht mehr eindeutig. Sie können damit einmal selbst experimentieren.
Nun zur Abfrage. Auch hier gibt es eine Schleife, die durch Eingabe von "END" beendet wird. Dabei wird in der Schleife der zu suchende Name mit INPUT eingelesen. Danach wird durch den Unterprogrammaufruf "lesen" die Datei "telefon" eröffnet. Nun wird in einer Schleife solange aus der Datei gelesen bis entweder eine Übereinstimmung zwischen n$ (Datei) und nam$(Anfragename) vorhanden ist oder **EOF(1)** einen Wert <> 0 liefert. EOF(1) sagt, wann das

```
"Interplan"
"(089) 1234066"
"Mayer"
"4711"
"END"
"0"
"Hugo"
"1234567"
"END"
"0"
```

Abb. 2.9.2
Telefon-Datensatz

Dateiende erreicht ist. EOF(1) liefert solange den Wert 0, wie Daten zum Einlesen vorhanden sind.
Hinter der Schleife wird dann abgefragt, ob EOF(1) einen Wert <> 0 lieferte, wenn ja, so wurde der Name in der Datei nicht gefunden. Wenn nicht, dann kann man die gefundene Telefonnummer ausgeben.
Diese hier verwendete Suche ist bei großen Dateien sehr zeitaufwendig, da der Computer im schlimmsten Fall die ganzen Daten einlesen müßte. Das ist so, als ob Sie im Telefonbuch immer bei Seite 1 beginnen würden, wenn Sie nach einer Nummer suchen. Man geht dabei so vor, daß man z.B. bei einem bestimmten Buchstaben anfängt zu suchen. Man könnte das hier auch einbauen, indem wir z.B. 26 Dateien anlegen, die für jeweils einen Anfangsbuchstaben zuständig sind und man dann nur dort sucht. *Abb. 2.9.2* zeigt übrigens den Inhalt der Datei "telefon", wie er nach einem kleinen Test aussehen kann.
Doch dazu gleich ein paar Aufgaben:
1. In der Datei "telefon" ist der Eintrag END 0 zu sehen. Schreiben Sie das Programm so um, daß bei Eingabe von END dieser Name nicht auch abgespeichert wird, und daß außerdem dann nicht noch nach einer Telefonnummer gefragt wird.
2. Versuchen Sie es einmal mit der alphabetischen Sortierung des Telefonbuches nach Anfangsbuchstaben. Dazu brauchen Sie einen neuen Befehl LEFT$(textvariable,1). Er isoliert Ihnen den ersten Buchstaben aus einer Textvariablen. Mit IF LEFT$(n$,1) = "A" THEN ... kann man dann Abfragen realisieren. Achten Sie darauf, daß man Anfangsbuchstaben groß und klein schreiben kann. Wählen Sie 26 Dateien z.B. mit den Namen "telefonA".."telefonZ". Sie müssen dazu die Unterprogramme "schreiben" und "lesen" mit Parametern versorgen.
In der Praxis gibt es natürlich noch viele andere Möglichkeiten, Dateien zu organisieren. So kann man zum Beispiel mit sogenannten relativen Dateien arbeiten. Dort kann man einen bestimmen Eintrag durch einen Index erreichen, ähnlich wie beim Zugriff auf ein Datenfeld. Dann legt man eine Indexdatei an, in der nur der Suchbegriff und die Position verzeichnet ist. Den eigentlichen Datensatz holt man dann erst, nachdem man die Indexdatei durchsucht hat.

# 3 Grafik mit dem Amiga-Basic

Im Einführungskapitel haben Sie schon mal kurz mit ein paar Grafik-Befehlen Kontakt bekommen.
Fangen wir doch gleich mal an. *Abb. 3.1* zeigt zwei neue Befehle. PSET und PRESET. Damit kann man einen Punkt setzen oder löschen.
Im Bild sind alle möglichen Varianten des Befehls abgedruckt. Die einfachste Form **PSET (x,y)** setzt einen Punkt an die Stelle x,y.
Dabei muß man darauf achten, daß beim Amiga die y-Achse von oben nach unten gezählt wird, wie bei den meisten Computer-Grafik-Bildschirmen und nicht wie in der Mathematik üblich von unten nach oben. Geben Sie einmal den Befehl:

**PSET (150,100)**

im Basic-Fenster ein. Ein einzelner Punkt taucht in etwa halber Bildhöhe auf.
PSET ermöglich es aber auch, eine andere Farbe zu verwenden. Dazu kann man den Farbcode, durch Komma getrennt, direkt dahinter schreiben. In unserem Standard-Fenster sind allerdings nur die Codes 0..3 zugelassen. 0 ist die Hintergrundfarbe, 1 ist weiß, 2 ist schwarz und 3 ist rot. Dabei sind die Einstellungen von den Grundeinstellungen des Amiga abhängig, denn man kann sie z.B. mit Preferences ändern.
Probieren Sie einmal den Befehl:

**PSET (150,100),3**

Ein roter Punkt wird hier gesetzt.
PRESET arbeitet im Prinzip gleich wie PSET, nur daß (wenn man keine Farbe angibt) die Hintergrundfarbe verwendet wird. Damit kann man also einen gesetzten Punkt wieder weglöschen.
Im Bild sehen Sie noch eine weitere Möglichkeit. Vor der Klammer kann zusätzlich das Wort **STEP** stehen. Wenn man dieses Wort einfügt, so beziehen sich die

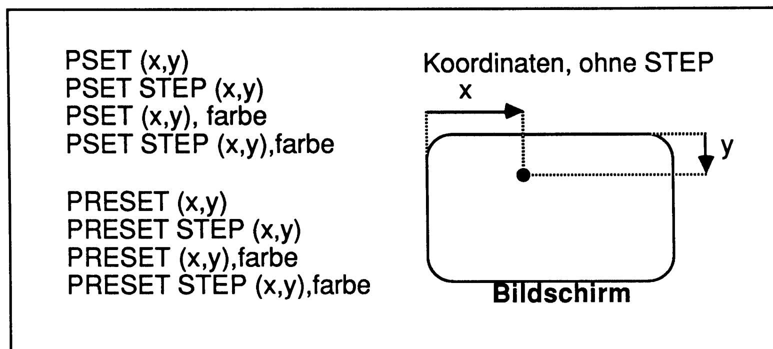



Abb. 3.1 PSET und PRESET-Befehle

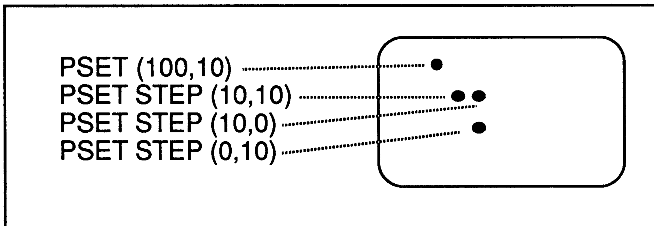



Abb. 3.2
Punkte auf dem
Bildschirm

angegebenen x,y-Koordinaten nicht mehr auf den linken oberen Bildschirmrand, sondern auf die zuletzt verwendeten Koordinaten. *Abb. 3.2* zeigt ein kleines Beispiel. Zunächst wird dort der Befehl PSET (100,10) gegeben. Den ersten Punkt muß man absolut festlegen, dann werden die weiteren Punkte relativ zu dieser Position angegeben. Mit PSET STEP (10,10) wird der nächste Punkt 10 Punkte nach rechts und 10 Punkte nach unten gezeichnet, usw.
Die relativen Koordinatenangaben sind ganz praktisch, wenn man z.B. Figuren zeichnet, denn man muß dann nur den Anfangspunkt, also den Start der Figur absolut angeben, alle weiteren Positionen sind dann unabhängig von der Position der gesamten Figur. Wenn Sie z.B. die Punktegruppe woanders im Bild sehen möchten, so müssen Sie nur den ersten Punkt durch neue x,y-Angaben woanders hinlegen. Der Rest des Programms bleibt unverändert.
Zwei weitere praktische Befehle sind: **CLS** löscht den Bildschirm und mit **COLOR vordergrundfarbe,hintergrundfarbe** kann man die aktuellen Zeichenfarben einstellen, so wie sie verwendet werden, wenn man keine Farbangabe bei den Befehlen macht.
Beispiel:

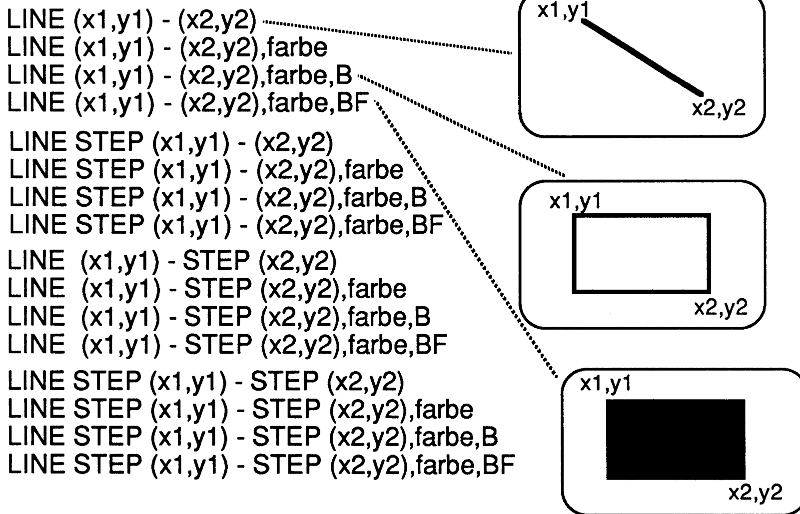
**COLOR 2,1**
**CLS**

Geben Sie diese beiden Befehle nacheinander ein (oder als Programm). Das Basic-Fenster ist danach weiß und die Schrift für die Kommandos ist schwarz. Mit COLOR 1,0 können Sie den alten Zustand wieder erreichen. Die definierte Hintergrundfarbe wird z.B. beim PRESET-Befehl verwendet.

*Abb. 3.3* zeigt die Möglichkeiten beim Linie-Befehl. Die einfachste Form zeichnet eine Linie beginnend bei x1,y1 nach x2,y2. Beispiel:

**LINE (0,0)-(639,199)**

Eine Diagonale wird quer über den Bildschirm gezeichnet. Achtung: Der Bildschirm ist zwar insgesamt 640 x 200 Punkt groß, dennoch paßt die Linie nicht ganz drauf. Das kommt daher, daß zunächst einmal die obere Menüleiste vom Platz abgeht. Ferner verliert man noch etwas Platz dadurch, daß man nicht ganz an den rechten Rand kommt (ca. 10 Punkte). Bei der Aufteilung des Schirms muß man darauf achten. Ferner kann man natürlich das Bildfenster mit Hilfe der Maus verkleinern, so daß die Fläche noch kleiner wird.
Es gibt übrigens einen Befehl, mit dem man die aktuelle Größe abfragen kann: WINDOW(2) liefert die aktuelle Breite und WINDOW(3) liefert die Höhe.
Mit **LINE (0,0)-(WINDOW(2),WINDOW(3))** wird die Diagonale immer genau passen.




Die Angabe STEP bewirkt, daß sich die nachfolgenden
Koordinaten auf die letzte aktuelle Bildschirmposition des Schreibzeigers beziehen

Abb. 3.3 Der LINE-Befehl

Line kann aber noch mehr. Man kann zum Beispiel die Zeichenfarbe angeben.
LINE (0,0)-(100,100),3 zeichnet eine rote Linie.
Außer Linien kann man mit LINE aber auch Rechtecke zeichnen. Dazu gibt man den Buchstaben b als weiteren Parameter an. Dann muß man allerdings die Zeichenfarbe immer mit angeben.

Beispiel:

```
FOR i = 1 TO 50
 LINE (100,10)-(i*4+100,i*2+10),1,B
NEXT i
```

Wenn Sie das Programm ausführen, erhalten Sie eine Reihe von ineinander geschachtelten Rechtecken.
Beim Programmieren von Figuren auf dem Amiga-Bildschirm muß man immer die aktuelle Auflösung im Auge behalten. Bei 640 x 200 Punkten gibt es in x-Richtung mehr als doppelt soviele Punkte wie in y-Richtung. Dadurch ist der Abstand in x-Richtung ca. halb so groß wie in y-Richtung. Wenn man daran nicht denkt bekommt man verzerrte Figuren. Aus diesem Grund ist hier die x-Koordinate mit einem doppelt so hohen Wert multipliziert wie die y-Koordinate. Tut man das nicht, sieht das Beispiel nicht symetrisch aus (probieren Sie es einmal).
Schließlich kann man beim Linien-Befehl noch den Buchstaben F anhängen. Er steht für Filled, also gefüllt. Das Rechteck wird dann mit der Zeichenfarbe ausgefüllt. Später, wenn wir Punktmuster definieren können, wird es auch mit einem solchen Muster ausgefüllt werden können.

Beispiel:

```
farbe = 0
FOR i = 50 TO 1 STEP -1
  LINE (100+i,10+i/2)-(i*4+100,i*2+10),1,BF
  farbe = farbe XOR 2
NEXT i
```

Das Programm zeichnet eine Art Pyramide, die aus abwechselnden Farbschichten besteht. *Abb. 3.4* zeigt das fertige Bild.

Ein weiteres interessantes Programm:

```
phi = 0
FOR x = 600 TO 10 STEP -5
  hoehe = SIN(phi) * 50
  LINE (x,x/4+20)-STEP(hoehe,hoehe/2),2,BF
  LINE (x,x/4+20)-STEP(hoehe,hoehe/2),1,B
  phi = phi + 3.141592/180 * 5
NEXT x
```

*Abb. 3.5* zeigt das Resultat. Durch das Programm entsteht eine Art 3D-Effekt. Der Linie-Befehl wird dazu zweimal verwendet. Das erste Mal zeichnet er die Fläche, beim zweiten Mal eine Umrandung. Hier kommt auch der STEP-Zusatz zum Einsatz. Dadurch spart man sich umfangreiche Berechnungen. Wenn Sie sich übrigens die erste "hoehe"-Angabe im Programm sparen, erhalten Sie einen 3D-Sinus, ähnlich zu den ersten Versuchen im Kapitel 2.
Der Trick bei der 3D-Darstellung besteht darin, daß das Bild von hinten nach vorne gezeichnet wird. Später gezeichnete Flächen überdecken ältere Flächen. Man nennt das auch Painters-Algorithmus, vom englischen Wort Painter im Sinne von Maler. Wenn man eine Landschaft zeichnet, beginnt man sinnvollerweise mit dem Hintergrund, dann zeichnet man nach und nach Objekte, die weiter vorne liegen. Dabei können weiter vorne liegende Objekt tiefer liegende verdecken. Würde man dagegen z.B. einen Baum im Vordergrund zuerst zeichnen und versucht dann später z.B. ein Haus dahinter zu zeichnen, so müßte man Teile des Hauses zwischen die Äste hineinmalen, was ungleich schwerer ist als der umgekehrte Weg.
Ein weiterer wichtiger Befehl ist der Kreis-Befehl. *Abb. 3.6* zeigt die verschiedenen Varianten des Befehls. Die einfachste Form lautet CIRCLE (x,y),r. Damit wird ein Kreis mit Mittelpunkt x,y und Radius r gezeichnet.
Probieren Sie dazu einmal folgendes Programm aus:

```
FOR r = 1 TO 50
  CIRCLE (200,100),r
NEXT r
```

Nach der Ausführung ist auf dem Bildschirm eine gefüllte Scheibe zu sehen. Dies ist insofern bemerkenswert, als bei anderen Computern die Scheibe meist von kleinen Unterbrechungen durchsetzt ist. Das Amiga-Basic macht es jedoch korrekt. Das kommt daher, daß sich hier der Radius auf die x-Achse bezieht. In y-Richtung wird verkleinert, so daß auch wirklich ein Kreis auf dem Bildschirm entsteht.
Die nächste mögliche Angabe beim Kreis ist die Farbe. Viel interessanter jedoch sind die nächsten beiden Angaben. Es handelt sich um den Start- und Endwinkel. Der Winkel wird dabei in Radiant angegeben, also 360 Grad entsprechen 2*3.141592... .

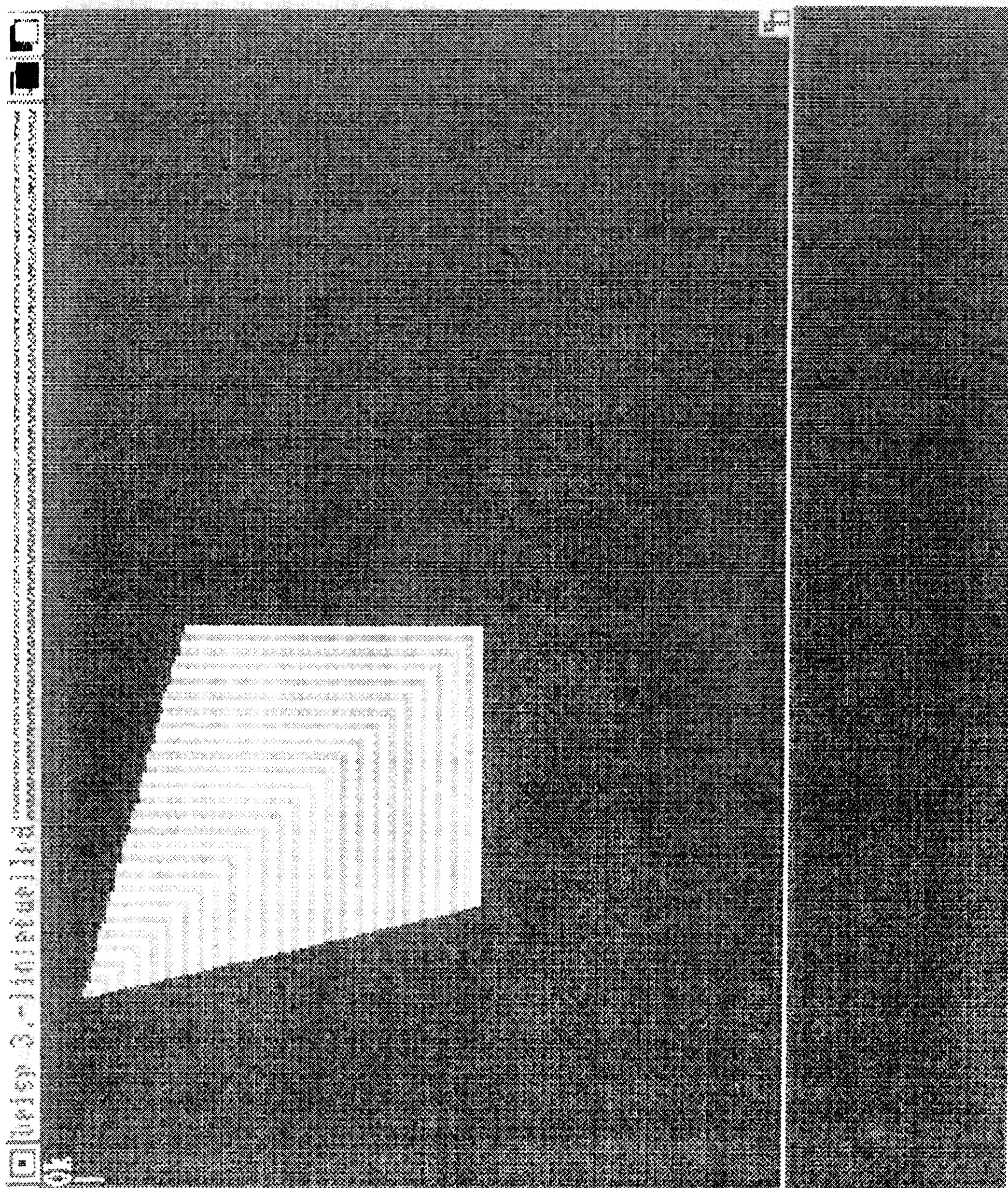


Abb. 3.4 LINE im Füllmode

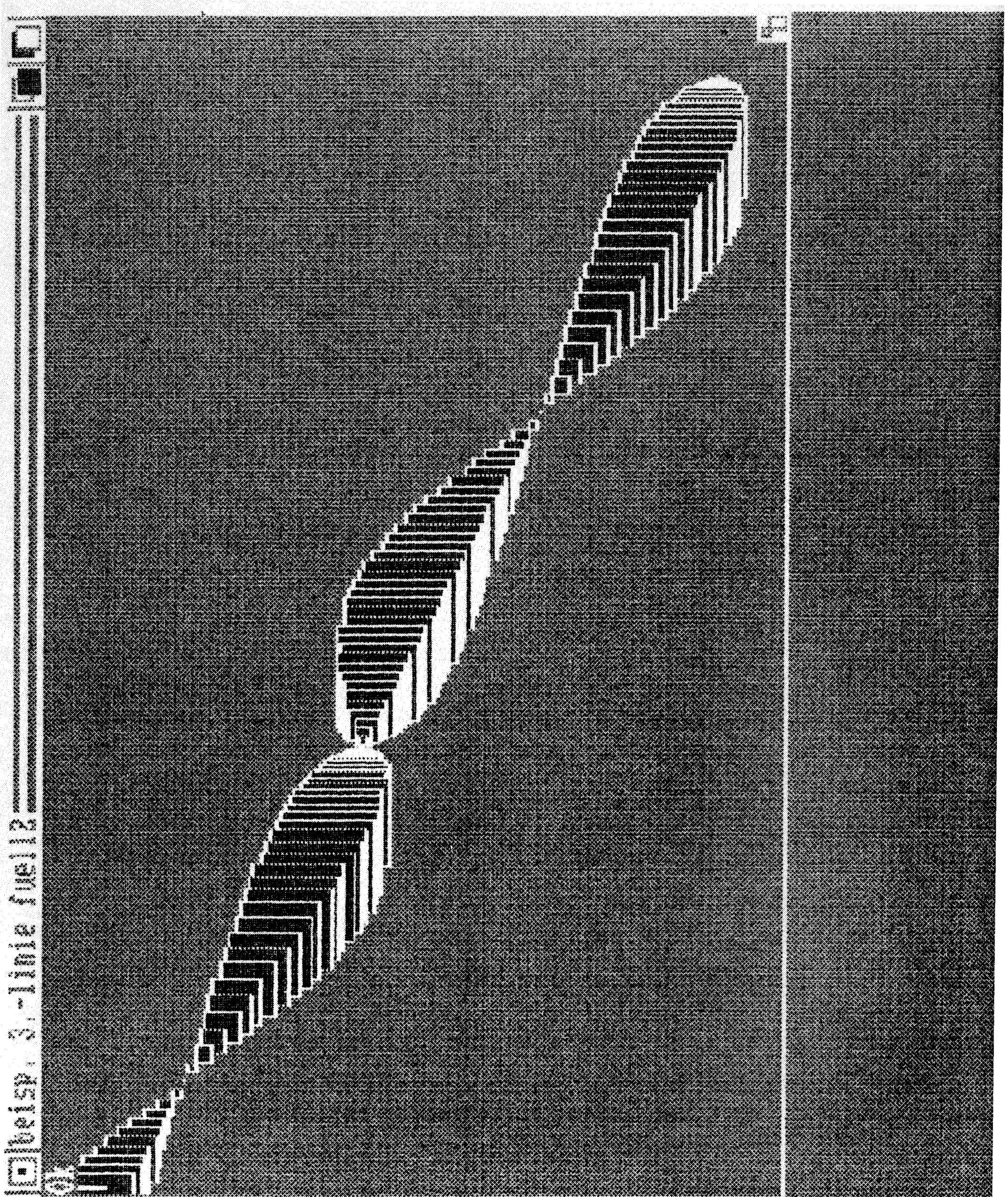
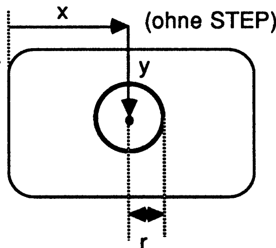



Abb. 3.5 Der Painters-Algorithmus

```
CIRCLE (x,y),r
CIRCLE (x,y),r,farbe
CIRCLE (x,y),r,farbe,startwinkel,endwinkel
CIRCLE (x,y),r,farbe,startwinkel,endwinkel,bildfaktor

CIRCLE STEP (x,y),r
CIRCLE STEP (x,y),r,farbe
CIRCLE STEP (x,y),r,farbe,startwinkel,endwinkel
CIRCLE STEP (x,y),r,farbe,startwinkel,endwinkel,bildfaktor
```



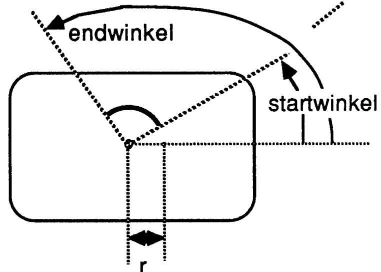




Abb. 3.6
Der CIRCLE-Befehl

Der Bildfaktor gibt das Seitenverhältnis für Ellipsen an, dabei muß auch das Bildschirmformat berücksichtigt werden.
Bei 640 x 200 Bildpunkten beträgt der Bildfaktor ca. 0.44, womit man einen Kreis erhält. Dieser Wert ist voreingestellt.
Der Wert errechnet sich aus dem Seitenverhältnis 2.25 : 1, nicht aus dem Teiler 640,200.

Beispiel:

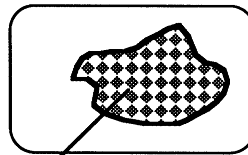
```
FOR r= 1 TO 50
  CIRCLE (200,100),r,3,0,3.141592*2/3
NEXT r
```

Dieses Programm zeichnet einen Kreisausschnitt.
Neben Kreisen kann man auch noch Ellipsen zeichnen. Dazu dient der letzte Parameter. Er gibt das Seitenverhältnis an. Durch diesen Parameter wird auch erreicht, daß Kreise auf dem Bildschirm auch bei unterschiedlicher Auflösung des Bildschirms kreisförmig erscheinen. Der Wert ist mit ca. 0.44 voreingestellt, was einem Seitenverhältnis von 2.25 : 1 entspricht. Dies entspricht nicht 640 : 200, denn dadurch ist nicht das Seitenverhältnis der Bildpunkte bestimmt.
Beispiel:

```
FOR aspekt = 0.11 TO 0.99 STEP 0.11
 IF aspekt = 0.44 THEN
  col = 1
 ELSE
  col = 3
 END IF
 CIRCLE (200,100),40,col,0,3.141592*2,aspekt
NEXT aspekt
```

PAINT (x,y)
PAINT (x,y),füllfarbe
PAINT (x,y),füllfarbe,randfarbe

PAINT STEP (x,y)
PAINT STEP (x,y),füllfarbe
PAINT STEP (x,y),füllfarbe,randfarbe



Die x,y-Koordinate, muß innerhalb des zu füllenden Gebiets liegen. Das Gebiet wird mit dem aktuellen Muster und der Füllfarbe bis zum Erreichen der Randfarbe ausgefüllt.

Abb. 3.7 Der PAINT-Befehl

Eine Reihe unterschiedlicher Ellipsen entsteht durch dieses Programm auf dem Bildschirm. Dabei wird der Kreis in weißer Farbe gezeichnet. Hinweis: die Abfrage IF aspekt = 0.44 THEN ist nicht ganz ohne Risiko. Denn intern werden Zahlen meist in binärer Schreibweise dargestellt. Bei der Umrechnung von Dezimal auf Binär können insbesondere bei Nachkommastellen Rechenungenauigkeiten auftreten. Das Amiga-Basic verhält sich zumindest bei diesem Programm friedfertig. Wenn man aber ganz sicher gehen will, so muß man z.B. schreiben IF (aspekt > 0.44 - 0.00001) AND (aspekt < 0.44 + 0.00001) THEN .... Also bitte merken: Abfragen auf Gleich sind gefährlich. Wird "aspekt" größer als Eins, so bezieht sich der Radius auf die y-Achse, probieren Sie es einmal aus.

Wenn man irgendwelche Flächen ausfüllen will, so kann man den Befehl PAINT verwenden. Die Syntax ist in *Abb. 3.7* dargestellt.
Die einfachste Form PAINT (x,y) füllt ein Gebiet mit der aktuellen Vordergrundfarbe (siehe COLOR-Befehl) auf. Dabei wird ausgehend von der Position x,y solange gefüllt, bis ein Rand der gleichen Farbe auftaucht.
Beispiel:

**CIRCLE (150,100),80**
**PAINT (150,100)**

Man erhält einen ausgefüllten Kreis.
Nun wollen wir den Kreis mit einer anderen Farbe ausfüllen. Dazu gibt man die Füllfarbe als nächsten Parameter an:

**CIRCLE (150,100),80**
**PAINT (150,100),3**

Nanu, der ganze Bildschirm ist jetzt rot, von Kreis keine Spur. Klar, denn CIRCLE verwendet, wenn man keine weitere Farbe angibt, die Füllfarbe auch als Randfarbe, da der Kreis weiß gezeichnet wurde, wird der Rand nicht gefunden, also eingeben:

**CIRCLE (150,100),80**
**PAINT (150,100),3,1**

Und schon hat man eine weiß umrahmte rote Kreisfläche. Damit hier nicht der Eindruck entsteht, daß man nur Kreise ausfüllen kann, hier ein komplexeres Beispiel, das in *Abb. 3.8* inclusive Programmergebnis abgebildet ist.

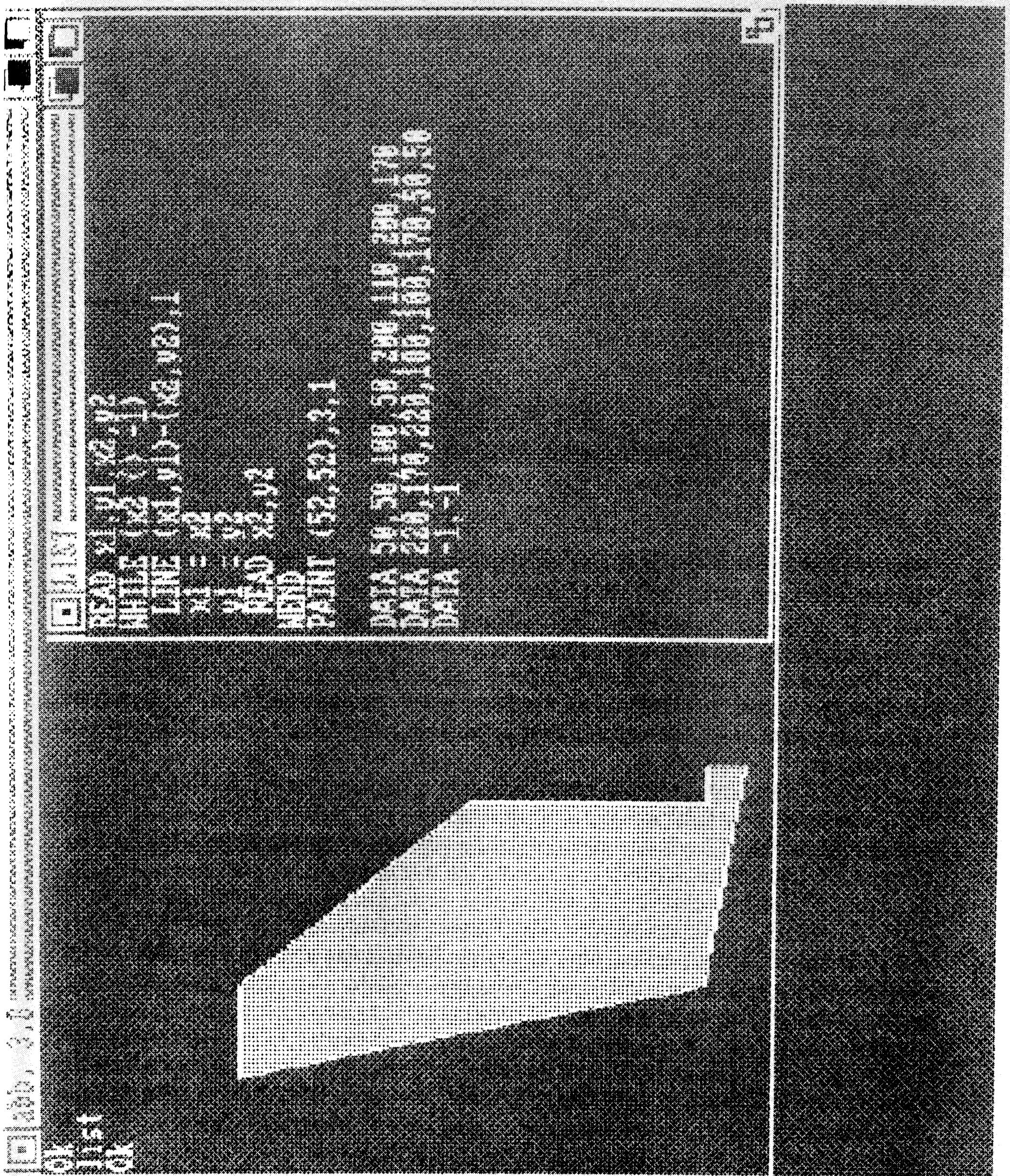



Abb. 3.8 Beispiel für PAINT

PATTERN linienmuster, füllmusterfeld
Das Füllmusterfeld muß ein Integerfeld sein (%-Zeichen anhängen) und die Anzahl der Elemente muß eine Zweierpotenz sein (zählen von 0 bis n-1)

Beispiel:

```
DIM muster%(3)
muster%(0) = &HF0F0
muster%(1) = &HF0F0
muster%(2) = &HF0F
muster%(3) = &HF0F
PATTERN &H5555,muster%
LINE (100,100)-(200,150),1,B
LINE (100,10)-(150,60),1,BF
```

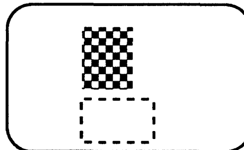


Abb. 3.9
Der PATTERN-Befehl

Achtung: Da der Bildschirm ein Seitenverhältnis von 2.25 : 1 besitzt, sind speziell die Linienmuster in horizontaler Richtung dichter als in vertikaler Richtung. Beim Füllmuster muß man das auch berücksichtigen.

Man kann aber beim Amiga-Basic nicht nur unicolor (einfarbig) ausgefüllte Flächen erzeugen, sondern auch Muster in Flächen ablegen.
*Abb. 3.9* zeigt den Befehl PATTERN. Dieser Befehl erlaubt es, zwei verschiedene Arten von Muster zu definieren. Der erste Parameter gibt das Muster an, wie es zum Zeichnen für alle Linien verwendet wird, der zweite dient zur Angabe des Musters für das Flächenfüllen. Um die Parameter verstehen zu können, müssen wir allerdings zuerst einen kleinen Ausflug in das sedezimale und binäre Zahlensystem machen.
Computer stellen Zahlen intern meist im binären Zahlensystem dar. Dieses Zahlensystem verwendet nur die Ziffern 0 und 1.
Die dezimale Zahl 12 wird im binären Zahlenssystem zum Beispiel als 1100 dargestellt. Die binäre Schreibweise brauchen wir um den Code für das Linienmuster zu berechnen. Z.B. würde eine gepunktete Linie dabei einfach als 101010101010 ... definiert werden. Der Pattern-Befehl verlangt eine 16-Bit-Zahl, das bedeutet 16 binäre Stellen. Nun kann man beim Basic aber Zahlen leider nicht direkt binär eingeben, das würde auch sehr unübersichtlich sein. Man könnte die binäre Zahl z.B. ins dezimale Zahlensystem umrechnen und dann als Parameter angeben. Das ist aber auch recht mühsam. Im Amiga-Basic gibt es noch eine andere Möglichkeit: Die Zahlen kann man im sedezimalen Zahlensystem angeben. Das sedezimale Zahlensystem wird übrigens manchmal auch fälschlicherweise als hexadezimales Zahlensystem bezeichnet, arbeitet mit 16 verschiedenen Ziffern pro Stelle, dabei zählt man: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,10,11,12...1F,20,..... .
Wie kann man nun binär dargestellte Zahlen einfach in sedezimal dargestellte Zahlen umwandeln. Dazu gibt es einen kleinen Trick. Man unterteilt die Binärzahl beginnend von rechts, also von der niederwertigen Stelle, in Vierergruppen auf. Jede dieser Viergruppen kann man jetzt unabhängig voneinander umrechnen. Dazu bedient man sich einer Tabelle:

| | |
|---|---|
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |

Beispiel: Die binäre Zahl 1011101011 soll in das sedezimale Zahlensystem umgerechnet werden.
Also 1. Unterteilen in Viergruppen:

10 1110 1011

Dann 2. Zuordnen der sedezimalen Ziffern:

2 E B

Die fertige Zahl lautet dann 2EB.
Sedezimale Ziffern gibt man beim Amiga-Basic durch voranstellen der Zeichen &H an.
Wir wollen jetzt einmal ein Linienmuster definieren. Es handelt sich um eine Strichpunktlinie:

-.-.-.-.

Die binäre Schreibweise ist dann z.B. 11010110101101011010...
Wir müssen nun das Ganze auf 16 Stellen begrenzen: 1101 0110 1011 0101, und in das sedezimale Zahlensystem übertragen: D6B5
Nun kann man den PATTERN-Befehl geben:

```
PATTERN &HD6B5
LINE (100,50)-(300,50)
LINE (300,20)-(300,180)
```

Wenn Sie genau hinsehen, erkennen Sie eine kleine Unregelmäßigkeit im Muster, auch ist es in x-Richtung schwer zu erkennen. Die Unregelmäßigkeit kommt durch die Begrenzung auf 16 Bit, denn das Muster wird durch das Basic einfach hinten wieder angehängt und das geht nicht ganz auf. Es treffen statt zwei Einsen drei aufeinander.
Also machen wir einen neuen Versuch, diesmal mit einem gröberen Muster: 1100 0011 1111 0000. Sedezimal lautet es: C2F0. Geben Sie diesen Wert anstelle von D6B5 im oberen Beispiel an. Diesmal verläuft das Muster ohne Störung und ist auch klar erkennbar.
Leider ist die y-Richtung dann sehr grob, so sollte man ggf. in x- und y-Richtung verschiedene Muster verwenden, wozu man aber jedesmal einen neuen PATTERN-Befehl geben muß.

Nun kann man beim Pattern-Befehl aber auch flächige Muster angeben. Dazu muß man ein Ganzzahlenfeld anlegen. Wir haben bisher immer mit Fließkommazahlen gearbeitet. Ganze Zahlen sind solche ohne Nachkommastellen. Außerdem sind sie in der Größe beschränkt. Eine Ganzzahl-Variable wird durch ein nachgestelltes Prozent-Zeichen definiert. Sie kann maximal eine 16-Bit-Zahl aufnehmen, die im Bereich -32768 .. +32767 liegen muß.
Ein Ganzzahlen-Feld wird durch die DIM-Anweisung definiert, wobei man dem Variablennamen ein Prozent-Zeichen anhängt. Beispiel: DIM feld%(8).
Beim PATTERN-Befehl wird das Muster in dieses Feld gelegt. Dabei werden für die horizontale Richtung immer 16-Bit verwendet und für die vertikale Richtung kann man eine Potenz von 2 als Zahl angeben. Das Muster kann also 1,2,4,8,16,32... usw. hoch sein.
*Abb. 3.10 (Farbtafel)* zeigt ein Beispielprogramm dazu.
Hier wird ein Kreis mit dem Muster ausgefüllt. Bei der Dimensionsangabe muß man darauf achten, daß man die Höhenangabe um Eins verringern muß, da das Feld bei 0 beginnend indiziert wird. Um solche Muster zu erstellen, zeichnet man sie am Besten auf ein Stück Karopapier. Man kann daraus leicht das Binärmuster abschreiben und umrechnen. Wem das zu kompliziert ist, der kann auch das Programm aus *Abb. 3.11* verwenden. Hiermit kann man mit Hilfe der Maus das Muster zeichnen.
Nach Start fragt das Programm zunächst nach der Höhe des Musters. Man muß dann

```
REM kleiner grafischer Editor fuer die Erzeugung von Pattern
REM Rolf-Dieter Klein
REM PATTEDI
hoehe = 3 : REM unzul.
WHILE (LOG(hoehe)/LOG(2)) <> INT(LOG(hoehe)/LOG(2)) AND (hoehe < 50)
 INPUT "Hoehe des Musters:",hoehe%
 hoehe = hoehe%
WEND
CLS
DIM muster%(hoehe%-1),std%(1)
std%(0) = &HFFFF
std%(1) = &HFFFF
gitter hoehe%
flag = 0
hebe = 1
WHILE flag = 0
 IF (MOUSE(0) < 0) THEN
  x% = MOUSE(1)
  y% = MOUSE(2)
  IF ((x% < 16*8) AND (y% < hoehe% * 4)) THEN
   h1% = x% \ 8
   v1% = y% \ 4
   IF hebe = 1 THEN
    defmode h1%,v1%,mode%
    hebe = 0
   END IF
```

```
   setzepunkt h1%,v1%,mode%
  ELSEIF (x% > 200) THEN
   flag = 1
  END IF
 END IF : REM Fehler im BASIC kein else
 IF (MOUSE(0) = 0) THEN
  IF hebe = 0 THEN
   hebe = 1
   PATTERN &HFFFF,muster%
   LINE (200,30)-STEP(100,50),1,bf
  END IF
 END IF
WEND
CLS
PATTERN &HFFFF,std%
FOR i=0 TO hoehe-1
 PRINT"&H";HEX$(muster%(i))
NEXT i
END

SUB gitter(hoehe%) STATIC
 FOR i = 0 TO 16
  LINE (i*8,0)-(i*8,hoehe%*4)
 NEXT i
 FOR i = 0 TO hoehe%
  LINE (0,i*4)-(16*8,i*4)
 NEXT i
END SUB

SUB setzepunkt(h%,v%,mode%) STATIC
 SHARED muster%(),std%()
 PATTERN &HFFFF,std%
 IF mode% = 1 THEN
  LINE (h%*8+1,v%*4+1)-STEP(6,2),3,bf
  muster = muster%(v%) OR 2^(15-h%)
 ELSE
  LINE (h%*8+1,v%*4+1)-STEP(6,2),0,bf
  muster = muster%(v%) AND 65535&-(2^(15-h%))
 END IF
 IF muster > 32767 THEN
  muster = muster - 65536&
 END IF
 muster%(v%) = muster
END SUB

SUB defmode(h%,v%,mode%) STATIC
 SHARED muster%()
 IF (muster%(v%) AND 2^(15-h%)) <> 0 THEN
  mode% = 0 : REM schon gesetzt dann loeschen
 ELSE
  mode% = 1 : REM setzen
 END IF
END SUB
```

Abb. 3.11
Pattern-Editor

eine Zweierpotenz eingeben. Das Programm erlaubt nur bis zur Höhe 32 eine Eingabe, wobei bei einem falschen Wert eine erneute Eingabe erforderlich ist. Dann wird ein Karo auf dem Bildschirm gezeichnet. Nun kann man mit der Maus in dieses Karo hineinklicken und so zeichnen. Ist ein Wert schon gesetzt, wenn man anklickt, so wird dieser Wert wieder gelöscht. Dabei bleibt der Mode (setzen oder löschen) solange aktiv bis man die Maustaste losläßt. Bei jedem Loslassen wird zum Test das Muster als kleine Fläche auf der rechten Bildschirmseite ausgegeben. Will man die Eingabe beenden, so klickt man z.B. das Muster auf der rechten Seite an. Alle Positionen mit x größer 200 werden als Ende interpretiert. Das Programm gibt am Schluß eine Liste der sedezimalen Codes aus, so wie sie der Pattern-Befehl versteht. Man kann das Programm auch noch erweitern, so daß man gleich ein kleines Basic-Programm erzeugt und die Datenliste auf Diskette abspeichert. Mit MERGE (siehe Basic-Handbuch) kann man ein solch erzeugtes Programm zu einem selbst geschriebenen dazumischen. In dem Programm kommen ein paar neue Befehle hinzu, insbesondere MOUSE(), auf die wir später nochmal zurückkommen und HEX$ zur Ausgabe in sedezimaler Schreibweise.
Ein weiterer wichtiger Befehl ist in *Abb. 3.12* dargestellt. Mit diesem Befehl kann man eine beliebige Fläche gestalten, ohne den PAINT-Befehl anwenden zu müssen. Der Befehl arbeitet dabei schneller als PAINT und hat auch sonst noch ein paar Vorteile. Beim PAINT tritt das Problem auf, einen Punkt in der auszufüllenden Fläche finden zu müssen. Das ist insbesondere bei berechneten Flächen nicht ganz einfach. Der AREA-Befehl definiert einfach die Umrandung des zu füllenden Gebiets, das anschließend mit AREAFILL gefüllt wird. Dabei liefert der erste AREA (x,y) - Befehl den Startpunkt. Jeder weitere AREA-Befehl fügt dann eine Eckkoordinate hinzu. Wenn man dann den Befehl AREAFILL 0 oder AREAFILL 1 gibt, so wird das Gebiet aufgefüllt. Wobei der letzte Punkt mit dem Startpunkt verbunden wird. Dabei wird mit AREAFILL 0 die aktuelle Schreibfarbe verwendet und mit AREAFILL 1 wird die jeweils im Bild vorhandene Farbe invertiert (es wird das Einerkomplement gebildet), wenn dorthin gemäß des Musters ein Punkt geschrieben werden sollte.
Beginnen wir mit einem einfachen Beispiel ohne PATTERN.

```
AREA (100,20)
AREA (300,30)
AREA (300,100)
AREA (120,70)
AREAFILL 0
```

AREA (x,y)
AREA STEP (x,y)

AREAFILL 0
AREAFILL 1

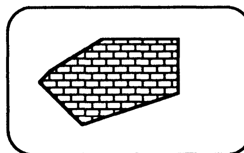


Abb. 3.12
Der AREA-Befehl

Die einzelnen Eckpunkte des zu füllenden Gebietes werden mit AREA definiert. AREAFILL füllt das Gebiet mit dem aktuellen Muster aus (ggf. Unicolor). Wird 0 als Parameter angegeben, so wird die aktuelle Farbe verwendet, bei 1 wird der Farbcode invertiert.

Auf dem Bild erscheint eine gefüllte Fläche. Wenn Sie einmal AREAFILL 1 ausprobieren, so wird die Fläche in Orange gezeichnet, da der Hintergrund zunächst mit dem Farbcode 0 (binär 00) = Blau beschrieben war. Der Code 1 (binär 01) entspricht Weiß, 2 (binär 10) entspricht Schwarz und 3 (binär 11) entspricht Orange. Aus dem binären Code 00 wird beim Invertieren (Einerkomplement) der binäre Code 11, was der Farbe Orange entspricht. Weiß wird zu Schwarz und umgekehrt. Orange wird zu Blau und umgekehrt. Die eingestellte Schreibfarbe bleibt unbeachtet.
Nun aber mit Muster:

```
DIM muster%(3)
muster%(0) = &HFFFF
muster%(1) = &HC00
muster%(2) = &HC00
muster%(3) = &HC00
LINE (0,70)-(220,100),1,BF
PATTERN &HFFFF,muster%
AREA (20,20)
AREA (100,120)
AREA (20,100)
AREAFILL 0
AREA (120,20)
AREA (200,120)
AREA (120,100)
AREAFILL 1
```

*Abb. 3.13 (Farbtafel)* zeigt das erzeugte Bild. Hier kann man deutlich den Unterschied zwischen AREAFILL 0 und AREAFILL 1 sehen. Bei Mode 0 ist die Fläche deckend. Bei Mode 1 wird der alte Untergrund sichtbar. Dabei werden die Farbcodes an den Stellen verknüpft, wo gemäß Muster sonst ein Bildpunkt gesetzt würde. Dort, wo im Untergrund die Farbe 1 (weiß) war, wird das inverse, also 2 genommen. Daher die schwarzen Linen. Wo im Untergrund die Farbe 0 war, wird der Wert 3 genommen, also orange. Mit diesem Mode kann man einen Transparent-Effekt erreichen.
Die Invertierung des Farbcodes bezieht sich auf das Binärmuster. Der Code 1 wird z.B. binär als 01 dargestellt. Nun invertiert man alle Bitstellen und erhält 10. Dabei ist das vom gewählten Bildschirmmode abhängig. Da wir hier im Moment nur mit 4 verschiedenen Farben arbeiten, der Farbcode also von 0 bis 3 reicht, werden nur zwei Binärstellen verwendet.
Bei 16 Farben wären es 4 Stellen und z.B. würde aus dem Farbcode 1 (0001) der Farbcode 14 (1110).
Die eingeführten Befehle sollen fürs erste genügen um ein paar anspruchsvolle Programme in den nächsten Unterkapiteln zu besprechen.

## 3.1 Diagramme und Funktionen

Die Darstellung von Diagrammen oder mathematischen Funktionen ist in der Praxis immer wieder wichtig. Dazu wollen wir einige praktische kleine Programme erstellen.

Das erste Programm soll sich mit Diagrammen beschäftigen. Bei Diagrammen sind die einzelnen Stützwerte vorgegeben und werden z.B. durch Meßwertaufnehmer oder per Hand geliefert. Wir wollen das Programm für eine Handeingabe bauen, so daß man z.B. statistische Daten direkt eingeben kann und dann das Diagramm dazu erhält.
Das Programm soll dabei automatisch den Bereich der eingegebenen Daten erkennen und so darstellen, daß alles auf den Bildschirm paßt.
Wie geht man bei der Programmentwicklung vor? Zunächst erstellt man ein Pflichtenheft. Das bedeutet, alle Eigenschaften des Programms werden schriftlich fixiert. Schreiben wir also mal die Eigenschaften unseres Programms nieder:
1. Eingeben der Diagramm-Daten
2. Darstellen des Diagramms

Das ist natürlich noch sehr grob und man muß nun das Pflichtenheft verfeinern:
1. Eingeben der Diagramm-Daten
1.1 Die Anzahl der Daten soll durch ein Endekennzeichen bestimmt werden.
1.2 Die Daten sollen in einem Feld gespeichert werden.
1.3 Nach der Eingabe soll eine Kontrolle durch Ausgabe der Daten möglich sein.
2. Darstellen der Diagramm-Daten
2.1 Ein Gitter soll gezeichnet werden
2.2 Maximal und Minimalwert soll im Gitter eingetragen werden.
2.3 Das Diagramm soll als Kurvenzug eingetragen werden

Man kann das Pflichtenheft nach belieben verfeinern. In diesem Stadium kann man noch leicht Änderungen durchführen. Als nächstes kann man einen groben Ablauf des Programms erstellen.
Das Programm könnte grundsätzlich so aussehen:

**einlesen**
**diagramm**

Damit hat man das Hauptprogramm schon fertig. So trivial das aussehen mag, wenn man so systematisch vorgeht, man nennt das Top-Down-Programmierung, kommt man rasch zu einem lauffähigen Programm.
Nun muß man die beiden Unterprogramme näher spezifizieren. Dabei muß man die Programmteile noch nicht unbedingt in der korrekten Basic-Schreibweise formulieren, sondern kann auch umgangsprachliche Elemente verwenden:

```
Unterprogramm einlesen:
wiederhole bis Ende der Eingabe
        lese Daten
        trage Daten in Feld ein
Ende Wiederholung
gib alle Elemente am Bildschirm aus
Ende Unterprogramm
```

Nun könnte man diesen Programmteil schon realisieren, doch wir machen erst mal mit dem Verfeinern beim Unterprogramm "diagramm" weiter:

```
Unterprogramm diagramm:
maximaminima min,max
gitterausgeben min,max
gibdiagrammaus min,max
Ende Unterprogramm
```

Hier sieht man, daß wir noch weitere Unterprogramme brauchen, "maximaxminima" soll aus dem Datensatz einen maximalen und minimalen Wert finden, "gitterausgeben " soll das Gitter zeichnen. Dazu braucht es auch min und max, denn wir wollten ja gemäß Pflichtenheft eine Angabe für den Maßstab. Das Unterprogramm "gitter" wird ferner noch die Zahl der Eingabeelemente brauchen, doch die können wir z.B. auch in einer globalen Variable speichern.
Das Unterprogramm "gibdiagrammaus" zeichnet schließlich das Diagramm. Es benötigt auch die Angaben min und max, denn die eingegebenen Diagrammwerte müssen auf die Bildschirmhöhe angepaßt werden.
Beginnen wir mit dem Unterprogramm "maximaminima". Hier kann man nun schon detailierter werden, da es sich um eine überschaubare Einheit handelt:

**Unterprogramm maximaminima:**
**nimm Anfangswerte für min und max an**
**wiederhole für alle Eingabedaten**
  **wenn Datenwert kleiner als altes min, dann**
    **min=Datenwert**
  **wenn Datenwert größer als altes max, dann**
    **max=Datenwert**
**Ende Wiederholung**
**Ende Unterprogramm**

Hier taucht noch ein Problem auf. Bei der Suche nach Maxima und Minima muß man die Variablen mit einem Anfangswert belegen. Dabei kann man für beide Werte z.B. den ersten Datenwert verwenden.
Nun das Unterprogramm "gitterausgeben":

**Unterprogramm gitterausgeben:**
**zeichne Anzahl horizontale Linien**
**zeichne vertikale Linien, z.B. 10 feste Anzahl**
**gib eine Legende für max und min aus.**
**Ende Unterprogramm**

Hier gibt es natürlich offene Fragen, denn unser Pflichtenheft war an dieser Stelle sehr ungenau. Dort stand nicht, wieviele Linien gezeichnet werden sollen. Wir wollen daher ergänzen:

2.1.1 Entsprechend der Anzahl der Daten, horizonale Linien
2.1.2 fest 10 Linien in vertikaler Richtung.

Bei dieser Angabe kann es auch noch zu Problemen kommen: Wenn sehr viele Daten eingegeben wurden, werden die Linien in horizontaler Richtung sehr groß. Dies überlasse ich jedoch den Lesern als Zusatzaufgabe.
Nun zur Ausgabe des Diagramms:

**Unterprogramm diagrammaus:**
**Setze Startpunkt fest**
**wiederhole für alle Daten**
  **zeichne Verbindung zum nächsten Datenpunkt**
**Ende Wiederholung**
**Ende Unterprogramm**

Nun kann man damit beginnen, das Programm zu codieren.
Fangen wir mit dem Hauptprogramm an. Man kann es praktisch so abschreiben wie in unserem groben Diagramm. Es besteht nur aus zwei Zeilen:

**einlesen**
**diagramm**

Damit man das eingegebene Programm möglichst frühzeitig testen kann, sollte man nun als nächstes die beiden Unterprogramme definieren. Dabei kann man in der ersten Testphase das zweite Unterprogramm leer lassen, das bedeutet, man schreibt nur **SUB diagramm STATIC** und in der nächsten Zeile **END SUB**. Den Programmteil "einlesen" wollen wir jetzt aber vollständig eingeben.
Vorher müssen wir noch überlegen, was wir als Endekriterium für das letzte Datum verwenden wollen. Man kann z.B. eine Zahl nehmen, die im normalen Datensatz nicht vorkommt, oder man liest eine Zeichenkette ein und wenn diese leer ist, ist die Eingabe beendet. Das wollen wir tun, da es neu ist. Man muß dann allerdings die Zeichenkette in eine Zahl umwandeln, doch dazu gibt es in Basic den VAL(textvariable) -Befehl. Eine leere Zeichenkette kann man abfragen, indem man auf zwei hintereinander geschriebene Anführungszeichen abfragt.
Nun gibt es noch eine Schwierigkeit. Wir haben noch kein Eingabefeld. Das müssen wir allerdings noch im Hauptprogramm definieren, und daher die Anweisung

```
DIM daten(300)
```

einfügen. Hier sollen einmal maximal 300 Datenelemente genügen. Auch über solche Grenzen muß man sich bei der Programmerstellung oft Gedanken machen. Damit wir das Feld im Unterprogramm ansprechen können müssen wir einen SHARED-Befehl verwenden. Der Feldname wird dann ohne Dimensionsangaben, aber mit Klammerauf, Klammerzu angegeben.
Wir brauchen noch eine globale Variable im Hauptprogramm, die die Anzahl der eingelesenen Daten beinhaltet. Nennen wir sie "zaehler". Im Hauptprogramm belegen wir sie einmal zur Sicherheit mit dem Wert 0, obwohl nach dem RUN immer alle Variable auf 0 gesetzt werden.
Die Variablen müssen wir ebenfalls in die SHARED-Anweisung schreiben. Unser Hauptprogramm sieht bis dahin so aus:

```
DIM daten(300)
zaehler = 0
einlesen
diagramm
```

Das Unterprogramm "einlesen" wie folgt:

```
SUB einlesen STATIC
SHARED daten(),zaehler
```

Jetzt können wir uns an die Einleseschleife machen. Wir brauchen im Unterprogramm eine Textvariable, die die Eingabe speichert und nennen sie eingabe$. Die Variable belegen wir mit einem Text, der nicht leer sein darf, damit die Schleife durchlaufen werden kann, also z.B.:

```
eingabe$="NIX"
```

Die Schleife beginnt dann mit:

```
WHILE eingabe$<>""
```

Nun können wir die INPUT-Anweisung schreiben. Schön wäre es noch, wenn man erfährt, welches Datum gerade ausgegeben wird. Dazu geben wir den Zählerstand zuerst mit einem Print aus und führen danach erst die INPUT-Anweisung aus.

```
PRINT "Eingabe von ";zaehler+1;". Datum:";
INPUT eingabe$
```

Dabei ist es wichtig, daß man den Strichpunkt hinter PRINT nicht vergißt, sonst beginnt INPUT in einer neuen Zeile. Der Zählerstand wird hier um Eins erhöht

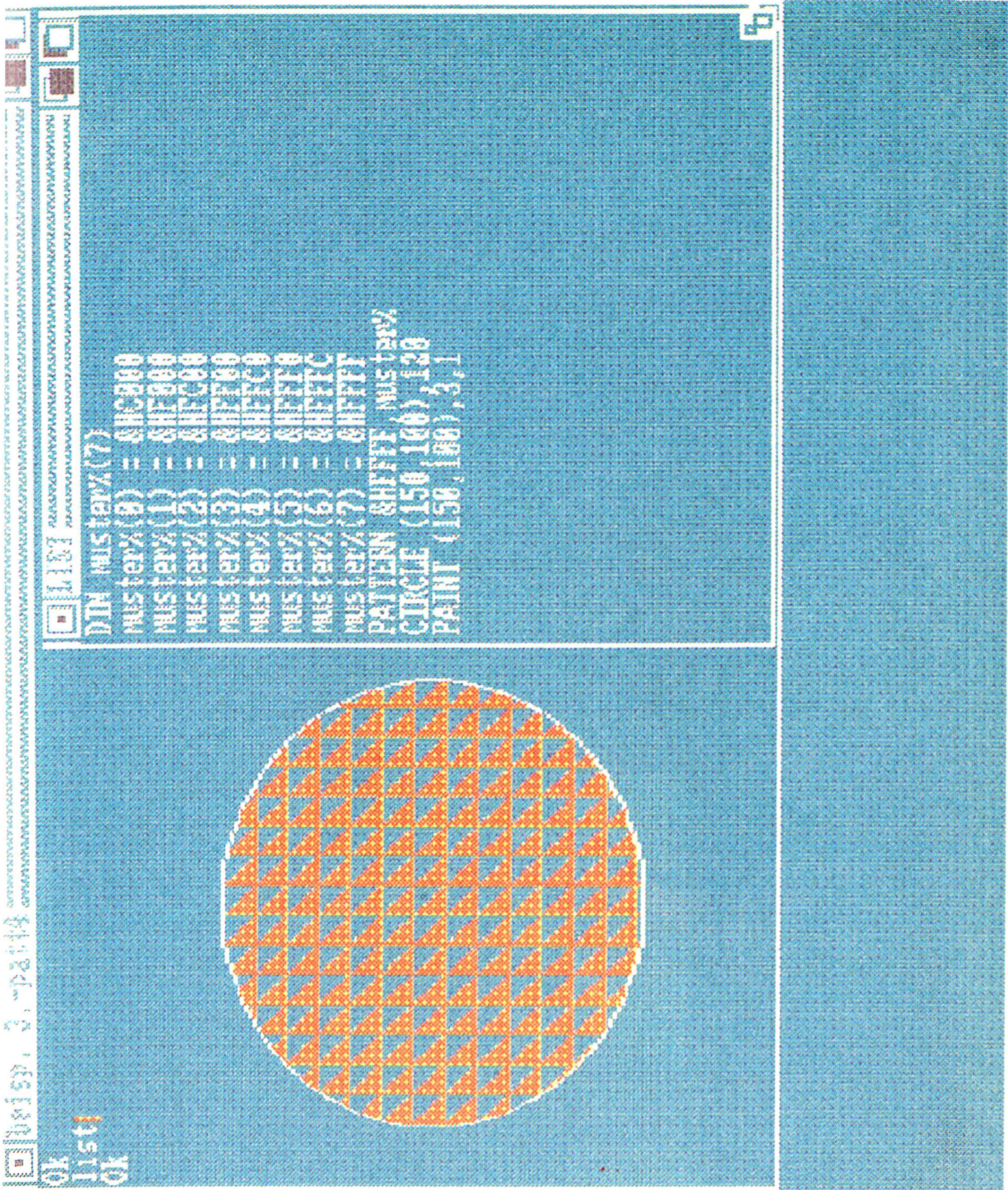



Abb. 3.10 Muster

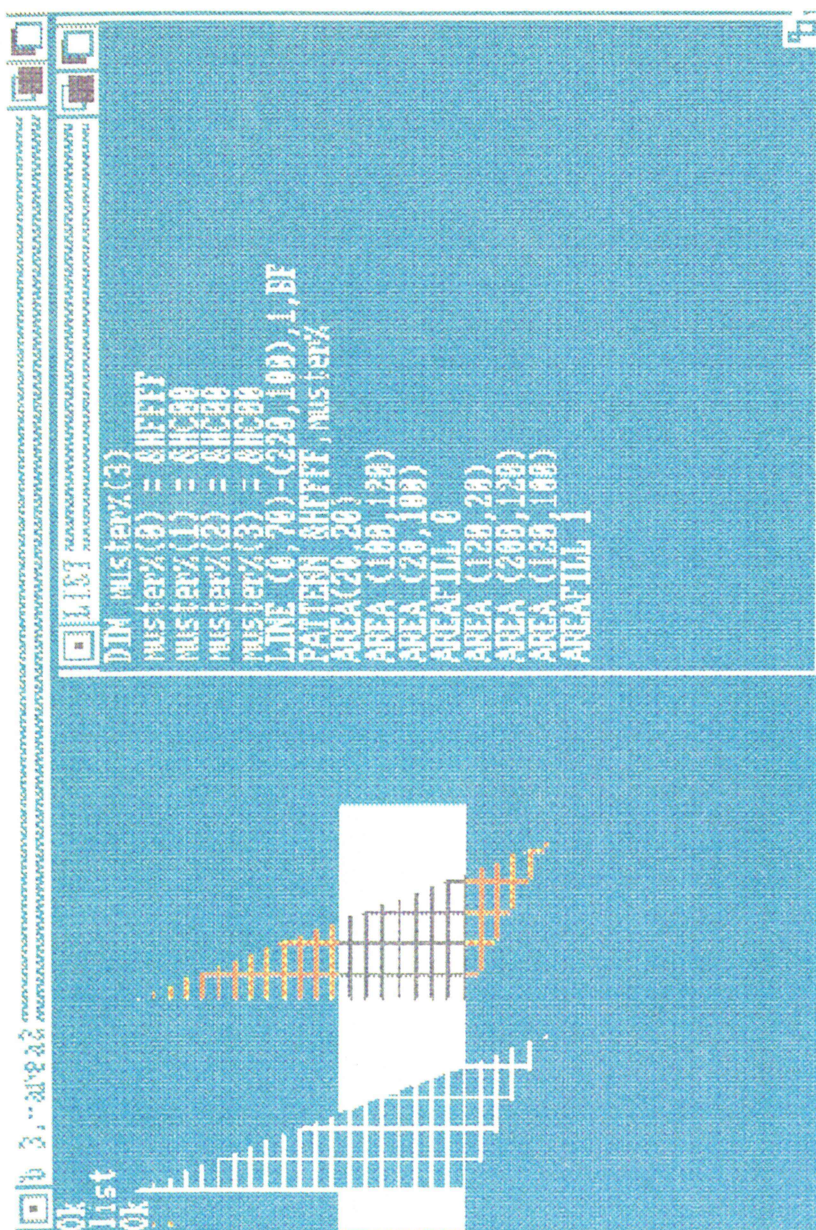



Abb. 3.13 AREAFILL

ausgegeben, da wir beim Index mit 0 zu zählen beginnen und in der Ausgabe 1 der erste Wert sein soll.
Das würde man allerdings schneller bei einem Test erfahren, den wir aber erst nach beenden des Programmteils durchführen wollen. Wenn der Eingabestring ungleich der leeren Eingabe ist, wollen wir den Wert abspeichern. Also

```
IF eingabe$ <> "" THEN
 daten(zaehler) = VAL(eingabe$)
 zaehler = zaehler +1
END IF
```

Nicht vergessen, daß man den Zähler hier erhöhen muß.
Nun ist die Schleife beendet und man schreibt:

```
WEND
```

Wenn man jetzt noch ein END SUB hinzufügt, kann man das Programm schon mal testen. Allerdings kann man hier nur Werte eingeben, die Kontrollausgabe fehlt noch. Das wollen wir jetzt nachholen:

```
FOR i=0 TO zaehler-1
 PRINT "Datensatz ";i+1; " ist ";daten(i)
NEXT i
```

Fügen Sie diesen Teil hinter der WEND-Anweisung unmittelbar vor dem END SUB ein.
Nun kann man das Programm wieder testen. Man müßte verschiedene Daten eintippen können und sobald man als letztes Datum nur die RETURN-Taste drückt, kommt die Liste der eingegebenen Daten, auf dem Bildschirm.
Aufgaben für den Leser:

1. Was passiert bei einer falschen Eingabe, z.B. wenn man einen Buchstaben eingibt.
2. Wie kann man das verhindern. Probieren Sie, das Programm dahingehend zu verbessern.
3. Was passiert, wenn man mehr als 300 Daten eingibt und wie kann man das verhindern.

Wir wollen jetzt aber mit dem zweiten Unterprogramm namens "diagramm " fortfahren. Dies ist nun auch wieder so ein Programm, das nur aus Unterprogrammaufrufen besteht. Man sollte diese Technik aber trotz des erhöhten Schreibaufwandes verwenden um Programme übersichtlich zu gestalten:

```
SUB diagramm STATIC
  maximaminima min,max
  gitterausgeben min,max
  gibdiagrammaus min,max
  END SUB
```

Da gibt es nicht viel zu testen, also machen wir erst mal weiter und schreiben das Unterprogramm "maximaxminima".

```
SUB maximaxminima(min,max) STATIC
```

Wir müssen als nächstes die Anfangswerte für min und max definieren und schreiben damit:

```
min = daten(0)
max = daten (0)
```

Nun kommt die Schleife dran. Also:

```
FOR i=1 TO zaehler -1
```

Doch Halt! Wir brauchen die Variable "zaehler" aus dem Hauptprogramm und natürlich auch "daten()". Daher müssen wir noch eine SHARED-Anweisung nach der Unterprogramm-Definition eintragen. Hätte man diese Anweisungen vergessen, so würde entweder das Programm nicht richtig arbeiten oder bestenfalls bekäme man eine Fehlermeldung (out of subscript) bei der Adressierung des Feldes, das bei fehlender SHARED-Anweisung als lokales Feld verstanden würde. Das Programm sieht bis hierher wie folgt aus:

```
SUB maximaxminima(min,max) STATIC
SHARED daten(),zaehler
min = daten(0)
max = daten (0)
FOR i=1 TO zaehler -1
```

Der Index i wurde hier beginnend bei 1 hingeschrieben, da der Index 0 schon behandelt wurde. Nun kann man die beiden Abfragen hinschreiben:

```
IF daten(i) < min THEN
  min = daten(i)
END IF
IF daten(i) > max THEN
  max = daten(i)
END IF
```

Damit kann man auch die NEXT-Schleife schliessen:

```
NEXT i
```

Wenn man nun mal probeweise eine Print-Anweisung einbaut, die die beiden Werte ausgibt, so kann man das Programm auch schon mal laufen lassen:

```
PRINT "test min:";min;" max:";max
END SUB
```

Geben Sie einmal ein paar Testdaten ein, z.B. 4, 3.1, -5, 9, 0, 1 usw.
Min und Maxwerte müssen am Schluß richtig ausgegeben werden. Wenn Sie die beiden anderen Unterprogramme noch nicht definiert haben, wird allerdings am Schluß des Programmlaufs eine Fehlermeldung vom Basic "undefined subprogramm" auftauchen, was aber noch nicht weiter stört.
Läuft der Programmteil mit verschiedenen Testsätzen, so kann man die PRINT-Anweisung wieder aus dem Unterprogramm herauslöschen. In der Praxis der Programmentwicklung verwendet man gerne solche zusätzlich eingebauten PRINT-Anweisungen, um Werte, die einem direkt nicht zugänglich sind, in der Programmentwicklungsphase kontrollieren zu können.
Beginnen wir das Unterprogramm "gitteraus". Hier können wir schreiben:

```
SUB gitterausgeben(min,max) STATIC
SHARED zaehler
```

Das Datenfeld "daten()" brauchen wir diesmal nicht in der SHARED-Anweisung anzugeben, da es in diesem Unterprogramm nicht verwendet wird.
Löschen wir erst einmal den Bildschirm.

```
CLS
```

Zeichnen wir dann die horizontalen Linien. Dabei soll der Linienabstand z.B. 18 betragen. Damit können wir den Bildschirm von y=0 bis y = 180 anfüllen.

```
FOR i = 0 TO 10
  LINE (0,i*18)-(600,i*18),2
NEXT i
```

Die Linien werden hier schwarz gezeichnet, um nicht zu stark zu stören. Dabei werden 11 Linien gezeichnet, um 10 Felder abzugrenzen. Die Koordinate x=600 soll der letzte verwendete Punkt sein. Mit einer einfachen Formel kann man erreichen, daß die Positionierung automatisch angepaßt wird:

index * 600 / (zaehler - 1)

Wenn der Index = 0 ist, so ist der Wert dieses Ausdrucks auch 0. Ist der Wert am Maximum (zaehler-1), so ist der Wert:

(zaehler-1)*600/(zaehler-1) also 600.

Nun die vertikalen Linien:

```
FOR i = 0 TO zaehler-1
 LINE (i*600/(zaehler-1),0)-STEP(0,180),2
NEXT i
```

Nach jedem dieser Schritte kann man das Programm schon testen, doch wir wollen auch den letzten Befehl noch eingeben um die Legende zu schreiben. Diese legen wir vielleicht am Besten an den unteren Bildschirmrand. Will man Text positionieren, so kann man den Befehl LOCATE verwenden. Er braucht zwei Parameter, die Zahl der Zeilen und die Zahl der Spalten, bezogen in Zeicheneinheiten:

```
LOCATE 23,3
PRINT "unterer Wert =";min; " oberer Wert = ";max;
END SUB
```

Wichtig ist, daß man einen Strichpunkt hinter PRINT schreibt, sonst kann es passieren, daß das Basic den Bildschirm hochschiebt.
Last not least das letzte Unterprogramm "diagrammaus".
Hier brauchen wir wieder das Datenfeld und den Zähler, also:

```
SUB gibdiagrammaus(min,max) STATIC
SHARED daten(),zaehler
```

Lesen wir als nächstes den Anfangspunkt ein .

```
y = daten(0)
```

Dann kann die Schleife beginnen:

```
FOR i = 1 to zaehler-1
```

Nun kann man die Linie zeichnen. Dabei muß man hier wieder ein bißchen rechnen. Zunächst muß man darauf achten, daß die y-Achse von oben nach unten geht. Dann soll y von 0 bis 180 reichen. Es gilt also 0 soll max entsprechen und 180 soll dem Wert min entsprechen. Man kann das so aufschreiben:

| | | |
|---|---|---|
| 0 | -> | max |
| 180 | -> | min |
| wert | -> | y |

Man darf diese Zuordnungen umformen:
paarweise subtrahieren:

| | | |
|---|---|---|
| 180-0 | -> | min-max |
| 180-wert | -> | min-y |

und teilen:

(180-wert)/(180) -> (min-y)/(min-max)

multiplizieren mit 180

180-wert -> (min-y)/(min-max)*180

Seiten herübertauschen (Addition, Subtraktion):

180-(min-y)/(min-max)*180 -> wert

Fertig ist die Formel. Ein kleiner Test:

y = min ergibt:
180-(min-min)/(min-max)*180= 180
y = max ergibt:
180-(min-max)/(min-max)*180 = 180 - 180 = 0

Und einbauen ins Programm:

**y1 = daten(i)**
**LINE((i-1)*600/(zaehler-1),180-(min-y)/(min-max)*180)-(i*600/(zaehler-1),180-(min-y1)/(min-max)*180),1**

Achtung, der LINE-Befehl muß in einer Zeile stehen.
Und das Ende des Unterprogramms:

**y = y1**
**NEXT i**
**END SUB**

Mit dem Befehl y = y1 wird der Endpunkt der Linie zum Anfangspunkt der nächsten Linie. Damit ist unser Programm fertig.
Wenn man noch verhindern will, daß das Programm mit der Meldung "OK" aufhört, kann man im Hauptprogramm den Befehl:

**WHILE MOUSE(0)=0**
**WEND**

einbauen. Erst wenn man die Maustaste drückt, wird dann das Programm beendet. *Abb. 3.1.1* zeigt das gesamte Programm. In *Abb. 3.1.2* ist ein mögliches Diagramm gezeigt.
Wenn man Funktionen darstellen will, so geht das mit unserem Programm ganz einfach. Dazu müssen wir Dank des modularen Konzepts nur ein Unterprogramm austauschen, nämlich das Unterprogramm "einlesen".
Der Datensatz wird dann einfach nicht per Hand, sondern durch eine Funktion bestimmt. Wir wollen mal z.B. die Funktion sin(phi)*cos(phi*11) darstellen. Dabei

```
REM Diagramm-Erstellung
REM Hauptprogramm
  DIM daten(300)
  zaehler = 0
  einlesen
  diagramm
  WHILE MOUSE(0) = 0
  WEND
END
REM Unterprogramme
SUB einlesen STATIC
 SHARED daten(),zaehler
 eingabe$="NIX"
 WHILE eingabe$<>""
  PRINT "Eingabe von ";zaehler+1;". Datum:";
  INPUT eingabe$
  IF eingabe$<>"" THEN
   daten(zaehler) = VAL(eingabe$)
   zaehler = zaehler + 1
```

```
  END IF
 WEND
 FOR i = 0 TO zaehler-1
  PRINT "Datensatz ";i+1;" ist ";daten(i)
 NEXT i
END SUB

SUB diagramm STATIC
 maximaminima min,max
 gitterausgeben min,max
 gibdiagrammaus min,max
END SUB

SUB maximaminima(min,max) STATIC
 SHARED daten(),zaehler
 min = daten(0)
 max = daten(0)
 FOR i=1 TO zaehler-1
  IF daten(i) < min THEN
    min = daten(i)
  END IF
  IF daten(i) > max THEN
    max = daten(i)
  END IF
 NEXT i
 REM : TEST TEST -> PRINT "test min:";min;" max:";max
END SUB

SUB gitterausgeben(min,max) STATIC
 SHARED daten(),zaehler
 CLS
 FOR i = 0 TO 10
  LINE(0,i*18)-(600,i*18),2
 NEXT i
 FOR i = 0 TO zaehler-1
  LINE(i*600/(zaehler-1),0)-STEP(0,180),2
 NEXT i
 LOCATE 23,3
 PRINT "unterer Wert =";min;" oberer Wert =";max;
END SUB

SUB gibdiagrammaus(min,max) STATIC
 SHARED daten(),zaehler
 y = daten(0)
 FOR i = 1 TO zaehler-1
  y1 = daten(i)
  REM Achtung: nachfolgender LINE Befehl muss in einer
  REM zusammenhaengenden Zeile eingetippt werden.
  LINE((i-1)*600/(zaehler-1),180-(min-y)/(min-max)*180)
       -(i*600/(zaehler-1),180-(min-y1)/(min-max)*180),1
  y = y1
 NEXT i
END SUB
```

Abb. 3.1.1 Diagramm-Erstellung

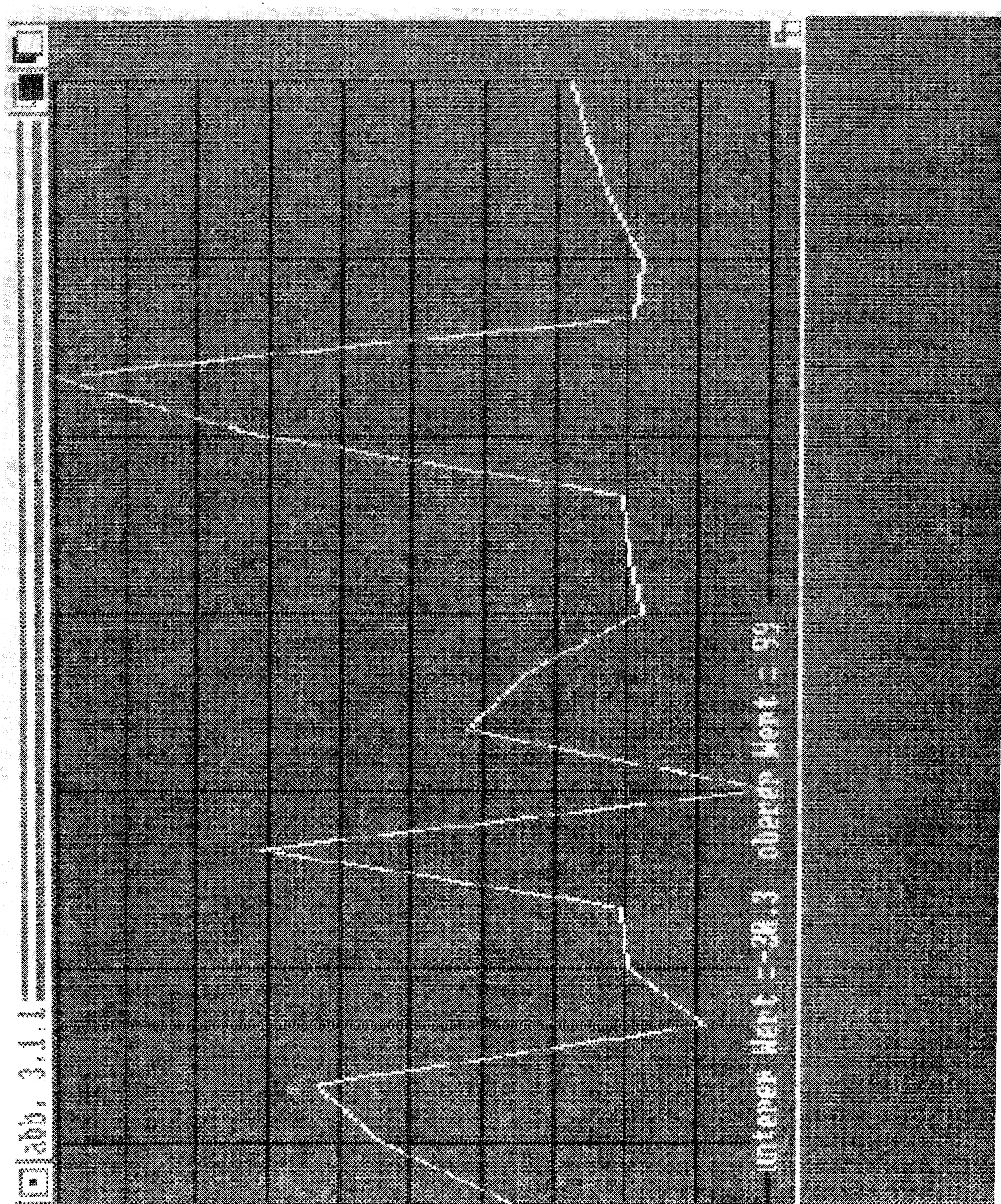



Abb. 3.1.2 Ein Diagramm

soll die Funktion im Bereich 0 bis 4*PI angezeigt werden. Den Winkel phi muß man dann aus dem zaehler berechnen.
Das Programm Einlesen sieht dann wie folgt aus:

```
SUB einlesen STATIC
SHARED daten(),zaehler
FOR zaehler = 0 TO 299
  phi = zaehler * 3.1412592 * 4 / 300
  daten(zaehler) = sin(phi)*cos(phi*11)
NEXT zaehler
zaehler = 300 : REM Anzahl speichern.
END SUB
```

Nach Start des Programms dauert es einen kleinen Moment bis alle 300 Punkte berechnet sind, dann erscheint das Gitter. Hier werden Sie feststellen, daß nun jeder zweite Punkt mit einer senkrechten Linie belegt wird, was jedoch zunächst nicht weiter stört. Die Kurve sollte aber ordnungsgemäß angezeigt werden.
Aufgaben:
1. Ändern Sie das Programm so, daß die Gitterlinien nie näher als 10 Punkte beisammenliegen, aber dennoch proportional der Anzahl der Daten sind.
2. Testen Sie einmal ein paar andere Funktionen, wie SQR(), EXP() usw.

## 3.2 Pie-Charts

Oft sieht man gerade bei Geschäftsgrafiken sogenannte Pie-Charts oder zu deutsch Tortenstückchen-Grafik. Damit kann man die Aufteilung von irgendwelchen Daten recht gut erkennen, z.B. Bevölkerungszahlen, Verkaufsstatistiken, Marktanteile usw. Im Gegensatz zu dem Diagramm-Programm werden hier die Daten nicht als absolute Zahlen angezeigt, sondern zueinander in Relation gesetzt.
Wir wollen ein Programm schreiben, das es erlaubt, Pie-Charts darzustellen. *Abb. 3.2.1* zeigt eine sehr einfache Form davon. Unser Programm muß einen Kreis teilen und die Sektoren darin gemäß der Datenverteilung eintragen. Der ganze Kreis repräsentiert 100%. Jeder eingezeichnete Sektor entspricht einer Prozentangabe davon.
Eingelesen werden soll ein Name und eine Wertangabe. Die Grafik soll anschließend entsprechend beschriftet werden.

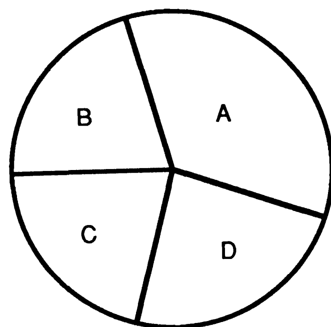



Abb. 3.2.1 Eine Pie-Chart

Beginnen wir mit dem Hauptprogramm. Wir brauchen ein Feld für die Namen und ein Feld für die Werte. In einer Variable "zaehler" speichern wir die tatsächlich eingegebene Zahl von Werten. Dann wird das Einleseprogramm aufgerufen und danach soll die Ausgabe erfolgen. Das Programm sieht dann so aus:

```
DIM namen$(20),werte(20)
zaehler = 0
einlesen
piechart
```

Dabei beschränken wir uns einmal auf ca. 20 Werte, mit mehr wird das Bild sehr unübersichtlich.

Doch nun zum Unterprogramm. Da wir im letzten Abschnitt beim Diagramm-Programm schon ein ähnliches Programm kennengelernt haben, können wir darauf zurückgreifen. Der wesentliche Unterschied ist hier, daß zusätzlich Namen eingelesen werden müssen. Ferner seien einmal nur positive Wertangaben erlaubt.

```
SUB einlesen STATIC
SHARED namen$(),werte(),zaehler
namen$(zaehler)="NIX"
WHILE namen$(zaehler)<>""
  INPUT "Text:";namen$(zaehler)
  IF namen$(zaehler) <> "" THEN
    werte(zaehler) = -1
    WHILE werte(zaehler)<0
      INPUT "Wert:";werte(zaehler)
      IF werte(zaehler)<0 THEN
        PRINT "nur Zahlen > 0 erlaubt"
      END IF
    WEND
     zaehler = zaehler + 1
     namen$(zaehler)="NIX"
  END IF
WEND
END SUB
```

Da wir diesmal gleich einen Text einlesen, kann man ihn auch dazu verwenden, um die Endeabfrage durchzuführen. Wichtig ist nur immer die Vorbelegung mit einem Text, wie z.B. "NIX", so daß die WHILE-Schleife nicht vorzeitig beendet wird. Es gibt natürlich auch noch andere Möglichkeiten, Sie können ja mal ein wenig experimentieren.

Doch nun zum Programmteil "piechart":

```
SUB piechart STATIC
summe sum
pieaus sum
END SUB
```

Um festzustellen, welchem Wert die 100% zukommen, müssen wir zunächste die Summe der Datenreihe berechnen. Dies geschieht wie folgt:

```
SUB summe(sum) STATIC
  SHARED werte(),zaehler
```

```
  sum = 0
  FOR i=0 TO zaehler-1
    sum = sum + werte(i)
  NEXT i
END SUB
```

In "sum" steht am Schluß die Summe aller Datenwerte.

Schon können wir beginnen, das eigentlich "pieaus"-Programm zu schreiben:

```
SUB pieaus(sum) STATIC
SHARED namen$(),werte(),zaehler
```

Als erstes löschen wir den Bildschirm und zeichnen einen Kreis, etwa in die Mitte des Bildschirms:

```
CLS
CIRCLE(300,100),80
```

Der Radius soll ca. 80 Punkte breit sein. Diesen Wert brauchen wir später zur Eintragung der Sektoren noch genau. Wir müssen nun "zaehler" Sektoren eintragen. Also brauchen wir eine FOR-Schleife. Wir brauchen noch einen weiteren Summenzähler, in dem wir die Eingabewerte akkumulieren.

```
sum2 = 0
FOR i=0 TO zaehler-1
```

Den Mittelpunkt fürs Linienzeichnen kennen wir, er liegt bei 300,100, doch nun kommt der schwierige Teil, die Bestimmung des Punktes auf dem Radius. Dabei ist es am einfachsten, den Winkel auszurechnen und daraus die Koordinate auf dem Kreis zu berechnen. In der Variablen "sum2" soll der aktuelle Datenwert aufaddiert werden.

Dabei gilt:

```
0       ->      0
sum     ->      2*PI
sum2    ->      winkel
```

also:

```
sum2/sum  -> winkel/2*PI
```

oder

```
2*PI*sum2/sum -> winkel
```

Damit können wir ins Programm eintragen:

```
sum2 = sum2 + werte(i)
winkel = 2*3.141592*sum2/sum
```

und jetzt brauchen wir noch eine Formel zur Bestimmung der x- und y-Koordinate auf dem Kreis. Sie lautet (nach Formelsammlung):

```
x = Radiusx*cos(winkel)+Mittex
y = Radiusy*sin(winkel)+ Mittey
```

Und damit lautet der Linien-Befehl:

```
LINE(300,100)-(80*COS(winkel)
        +300,-80*0.44*SIN(winkel)+100)
```

Die Zahl 0.44 kommt daher, daß in y-Richtung der voreingestellte Aspect (siehe Kreis-Befehl) verwendet werden muß, da in y-Richtung weniger Punkte auf dem Bildschirm vorhanden sind. Das negative Vorzeichen bei y muß man hinschreiben, da die y-Achse nach unten zeigt.

Mit

**NEXT i**
**END SUB**

ist das Programm lauffähig. Sie können das Programm schon mal testen.
Nun fehlt noch die Beschriftung. Da man Texte leider nicht so leicht Punktgenau positionieren kann, wollen wir den Text nicht in die Sektoren schreiben, sondern rechts daneben. Eine Linie soll dann zu den Sektoren zeigen.
Dazu schreiben wir vor das Unterprogrammende folgende Zeilen:

```
FOR i=0 TO zaehler-1
 LOCATE i+1,60
 PRINT name$(i)
NEXT i
```

Damit wird der Text ausgegeben. Mit LOCATE kann man den Text positionieren, allerdings wird bei den Angaben mit Zeicheneinheiten gerechnet. Achtung, beide Parameter (Zeile und Spalte) müssen größer gleich 1 sein.
Nun zu den Linien.
Bei der Berechnung des Winkels wollen wir einen Punkt in dem Sektor finden und dazu kann man z.B. den Mittelwert zweier Winkel verwenden.

```
sum2 = 0
FOR i=0 TO zaehler-1
 sumold = sum2
 sum2 = sum2 + werte(i)
 winkel = 2*3.141592*(sumold+sum2)/(sum*2)
 LINE(470,i*8+4)-( 40*COS(winkel)
      +300,-40*0.44*SIN(winkel)+100),3
NEXT i
```

Damit ist das Programm fertig.
*Abb. 3.2.2* zeigt das komplette Listings des Programms und *Abb.3.2.3* einen Schirmausdruck.
Aufgaben:
1. Versuchen Sie, die Texte in größerem Abstand auszugeben.
2. Die Verteilung der Texte ist nicht optimal. Versuchen Sie, das Programm so zu verbessern, daß sich die Linien nicht kreuzen. Dazu müßten ggf. die Texte umsortiert werden (Achtung, die Aufgabe ist nicht leicht!).

```
REM PIE-CHARTS
REM Rolf-Dieter Klein 870514
DIM namen$(20),werte(20)
zaehler = 0
einlesen
piechart
END
REM Unterprogramme

SUB einlesen STATIC
 SHARED namen$(),werte(),zaehler
 namen$(zaehler)="NIX"
```

```
 WHILE namen$(zaehler)<>""
  INPUT "Text:";namen$(zaehler)
  IF namen$(zaehler)<>"" THEN
   werte(zaehler) = -1: REM startwert
   WHILE werte(zaehler)<0
    INPUT "Wert:";werte(zaehler)
    IF werte(zaehler)<0 THEN
     PRINT "nur Zahlen > 0 erlaubt"
    END IF
   WEND
   zaehler = zaehler + 1
   namen$(zaehler)="NIX"
  END IF
 WEND
END SUB

SUB piechart STATIC
 summe sum
 pieaus sum
END SUB

SUB summe(sum) STATIC
 SHARED werte(),zaehler
 sum = 0
 FOR i=0 TO zaehler-1
  sum = sum + werte(i)
 NEXT
END SUB



SUB pieaus(sum) STATIC
 SHARED namen$(),werte(),zaehler
 CLS
 CIRCLE(300,100),80
 sum2 = 0
 FOR i=0 TO zaehler-1
  sum2 = sum2 + werte(i)
  winkel = 2*3.141592*sum2/sum
  LINE(300,100)-(80*COS(winkel)+300,-80*.44*SIN(winkel)+100)
 NEXT i
 FOR i=0 TO zaehler-1
  LOCATE i+1,60
  PRINT namen$(i)
 NEXT i
 sum2 = 0
 FOR i=0 TO zaehler-1
  sumold = sum2
  sum2 = sum2 + werte(i)
  winkel = 2*3.141592*(sumold+sum2)/(sum*2)
  LINE(470,i*8+4)-(40*COS(winkel)+300,-40*.44*SIN(winkel)+100),3
 NEXT i
END SUB
```

Abb. 3.2.2
Das Pie-Chart-Programm

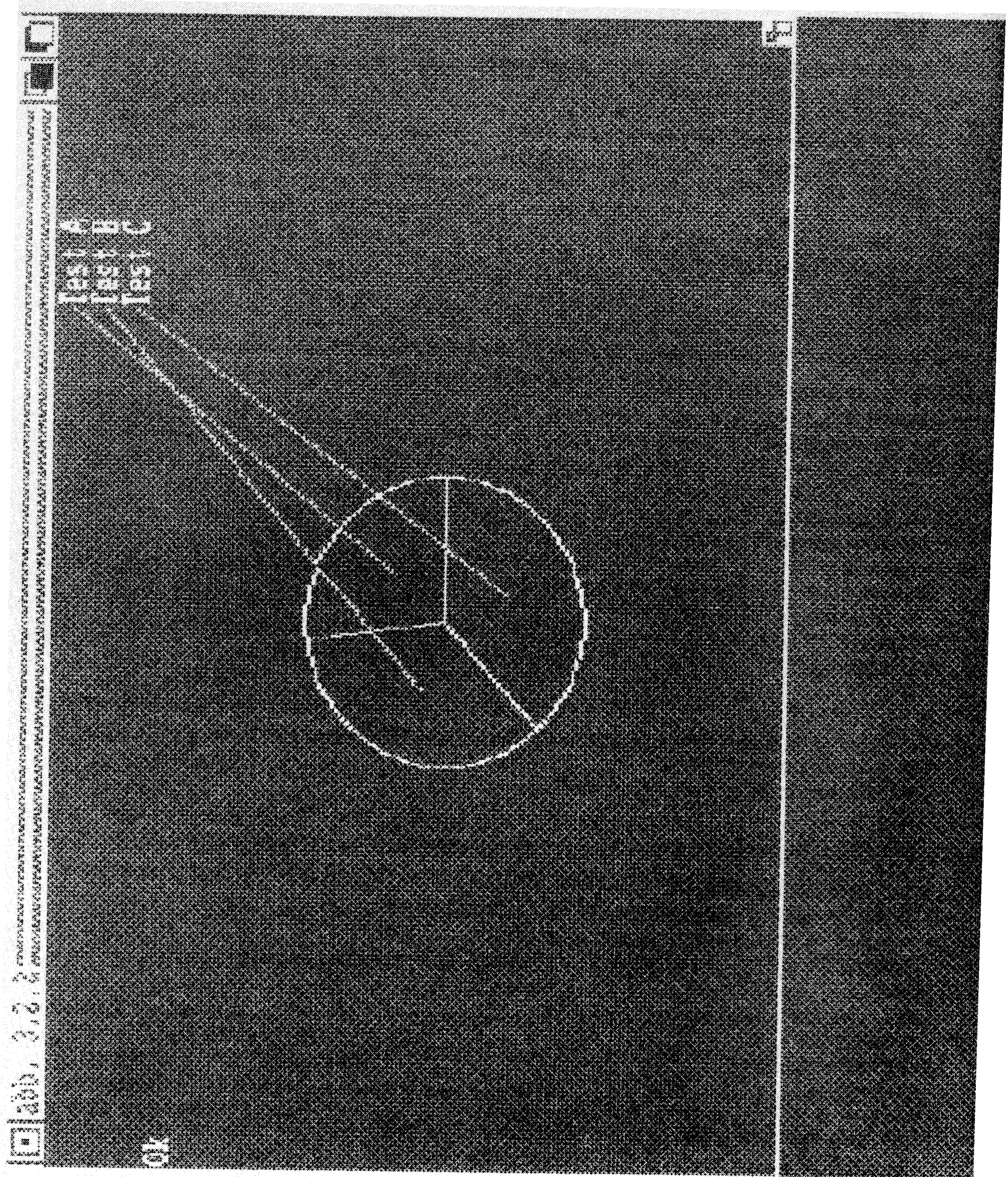



Abb. 3.2.3 Ausgabe des Pie-Chart-Programms

3. Füllen Sie Flächen mit einem Muster aus. Jede Fläche sollte ein anderes Muster bekommen (Hinweis: Verwenden Sie PAINT, und als Paint-Punkt den Endpunkt der Text-Zeiger-Linie, wie wir ihn im Unterprogramm "pieaus" schon berechnet haben). Füllen Sie zunächst einmal ohne Muster und bauen dann später eigene Muster hinzu.

## 3.3 3D-Säulengrafik

Wenn man statistische Daten besonders hübsch darstellen will, so empfiehlt es sich, ein Säulendiagramm zu verwenden. Dabei werden, wie schon der Name sagt, einzelne Säulen verwendet, um die Werte zu repräsentieren. Diese Säulen sind aber dreidimensional dargestellt. Solch ein Programm wollen wir schreiben. Dazu fangen wir einmal mit dem Pflichtenheft an:

1. Einlesen der Daten, Endekennung durch RETURN
2. Ausgabe als Säulen. Die Höhe entspricht dabei den eingegebenen Werten.

Das Einleseprogramm können wir praktisch vom Programm aus Kapitel 3.1 verwenden, genauso wie die Maximaminima-Suche, die wir hier auch wieder brauchen werden.
Das Hauptprogramm sieht dann so aus:

```
DIM daten(300)
zaehler = 0
einlesen
saeulen
WHILE MOUSE(0)=0
WEND
END
```

Bis auf den Aufruf von "saeulen" entspricht es genau dem Programm aus Abb. 3.1.1. Das Unterprogramm "einlesen" ist identisch, und wir brauchen es nicht neu hinzuschreiben.
Damit können wir das Unterprogramm "saeulen" formulieren:

```
SUB saeulen STATIC
  maximaminima min,max
  gibsaeulenaus min,max
END SUB
```

Das Unterprogramm "maximaminimax" kann ebenfalls aus dem alten Programm direkt übernommen werden, und wir widmen uns jetzt der Säulendarstellung, also dem Unterprogramm "gibsaeulenaus":

```
SUB gibsaeulenaus(min,max) STATIC
 SHARED daten(),zaehler
 CLS
```

*Abb. 3.3.1* zeigt das Schema der Säule. Dabei sollen die Seitenflächen etwas dunklere Töne besitzen als die Vorderseite. Man könnte dies zum Beispiel durch ein Füllmuster erreichen, aber der Amiga hat da noch andere Möglichkeiten. Wir verwenden einen neuen Befehl: PALETTE nummer,rot,gruen,blau. Damit kann man die Farbe für einen bestimmten Farbcode (bei uns 0..3) einstellen. Jeder der Farbwerte kann im Bereich 0..1.0 eingestellt werden, wobei der Amiga 16 verschiedene Stufen pro Farbcode zuläßt.

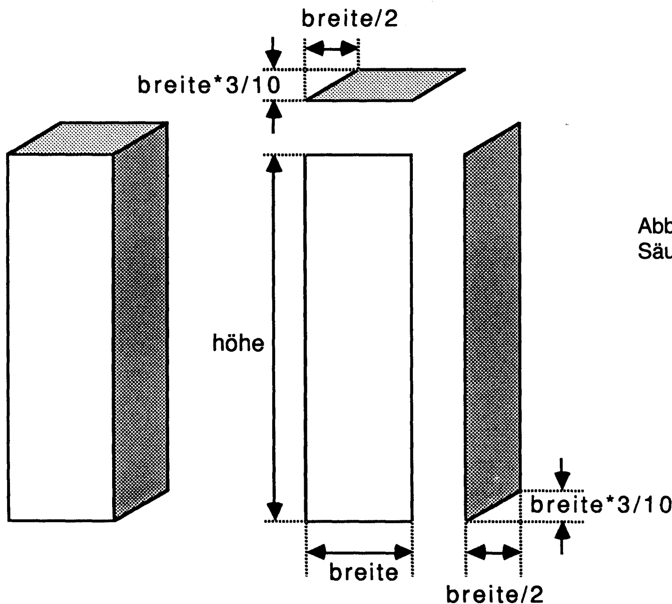



Abb. 3.3.1
Säulen – Schema

Der Code 1 ist mit der Farbe Weiß (1.0,1.0,1.0) schon voreingestellt, wir programmieren daher nur die anderen Farben neu:

**PALETTE 2,0.5,0.5,0.5**
**PALETTE 3,0.8,0.8,0.8**

Dadurch bekommen wir zwei weitere Graustufen. Die Farbe 3 verwenden wir dann für die obere Fläche und die Farbe 2 für die Seitenfläche.
Nun zur Ausgabe der Säulen. Die Breite der Säulen errechnet sich aus der gesamten Anzahl der Daten. Dabei soll zwischen den Säulen ein wenig Platz bleiben. Ferner muß berücksichtigt werden, daß rechts neben der letzten Säule auch Platz sein muß, daher wird hier durch "zaehle+1) dividiert.

**breite = 1/2*600/(zaehler+1)**

Dabei ist "breite" die Breite der Säule ohne 3D-Effekt. Der Platz für den 3D-Teil ist in der obigen Formel mit berücksichtigt. Als Zwischenraum bleibt dann 1/4 des Abstands der Säulen. Nun zur Ausgabeschleife:

**FOR i=0 TO zaehler-1**

Wir berechnen zunächst noch die Höhe der Säulen. Dabei muß man berücksichtigen, daß durch die 3D-Darstellung Platz über der Säule gebraucht wird. Von der verfügbaren Gesamthöhe (180 Punkt) muß man hier noch 3/10*breite abziehen.

**y1 = daten(i)**
**maxhoehe = 180-3/10*breite**
**hoehe = (min-y1)/(min-max)*maxhoehe**

Die Berechnung von "maxhoehe" ergibt immer den gleiche Wert, man kann diese Formel auch vor die Schleife setzen, um Rechenzeit zu sparen. Nun kann man die einzelnen Flächenelemente ausgeben, doch dazu braucht man noch die x-Koordinate für den linken unteren Punkt der Säule. Dabei muß man hier so tun, als ob man eine

Säule mehr hätte, so daß die letzte Säule noch rechts aufs Bild paßt. Daher wird hier durch "zaehler" und nicht durch "zaehler-1" dividiert.

**x = i*600/zaehler**

Mit COLOR kann man nun die jeweilige Farbe bestimmen und mit AREA und AREAFILL die Fläche angeben, beginnen wir mit dem Grundelement:

```
COLOR 1,0
AREA (x,180)
AREA (x+breite,180)
AREA (x+breite,180-hoehe)
AREA (x,180-hoehe)
AREAFILL
```

Dann die Seitenflächen:

```
COLOR 2,0
AREA (x+breite,180)
AREA (x+breite+breite/2,180-breite*3/10)
AREA (x+breite+breite/2,180-hoehe-breite*3/10)
AREA (x+breite,180-hoehe)
AREAFILL
COLOR 3,0
AREA (x,180-hoehe)
AREA (x+breite,180-hoehe)
AREA (x+breite+breite/2,180-hoehe-breite*3/10)
AREA (x+breite/2,180-hoehe-breite*3/10)
AREAFILL
```

schließlich noch:

```
NEXT i
END SUB
```

Und fertig ist unser Programm. *Abb. 3.3.2* zeigt das komplette Listing und in *Abb. 3.3.3* ist ein Beispieldiagramm dargestellt.

```
REM 3D-Saeulendiagramm
REM Hauptprogramm
  DIM daten(300)
  zaehler = 0
  einlesen
  saeulen
  WHILE MOUSE(0) = 0
  WEND
END
REM Unterprogramme
SUB einlesen STATIC
 SHARED daten(),zaehler
 eingabe$="NIX"
 WHILE eingabe$<>""
  PRINT "Eingabe von ";zaehler+1;". Datum:";
  INPUT eingabe$
  IF eingabe$<>"" THEN
   daten(zaehler) = VAL(eingabe$)
   zaehler = zaehler + 1
  END IF
 WEND
 FOR i = 0 TO zaehler-1
```

```
  PRINT "Datensatz ";i+1;" ist ";daten(i)
 NEXT i
END SUB

SUB saeulen STATIC
 maximaminima min,max
 gibsaeulenaus min,max
END SUB

SUB maximaminima(min,max) STATIC
 SHARED daten(),zaehler
 min = daten(0)
 max = daten(0)
 FOR i=1 TO zaehler-1
  IF daten(i) < min THEN
    min = daten(i)
  END IF
  IF daten(i) > max THEN
    max = daten(i)
  END IF
 NEXT i
 REM : TEST TEST -> PRINT "test min:";min;" max:";max
END SUB


SUB gibsaeulenaus(min,max) STATIC
 SHARED daten(),zaehler
 PALETTE 2,.5,.5,.5
 PALETTE 3,.8,.8,.8
 CLS
 breite = 1/2*600/(zaehler+1)
 maxhoehe = 180-3/10*breite
 FOR i=0 TO zaehler-1
  y1 = daten(i)
  hoehe = (min-y1)/(min-max)*maxhoehe
  COLOR 1,0
  x = i*600/(zaehler)
  AREA (x,180)
  AREA (x+breite,180)
  AREA (x+breite,180-hoehe)
  AREA (x,180-hoehe)
  AREAFILL
  COLOR 2,0
  AREA (x+breite,180)
  AREA (x+breite+breite/2,180-breite*3/10)
  AREA (x+breite+breite/2,180-hoehe-breite*3/10)
  AREA (x+breite,180-hoehe)
  AREAFILL
  COLOR 3,0
  AREA(x,180-hoehe)
  AREA(x+breite,180-hoehe)
  AREA(x+breite+breite/2,180-hoehe-breite*3/10)
  AREA(x+breite/2,180-hoehe-breite*3/10)
  AREAFILL
 NEXT i
END SUB
```

Abb. 3.3.2 3D-Säulenprogramm

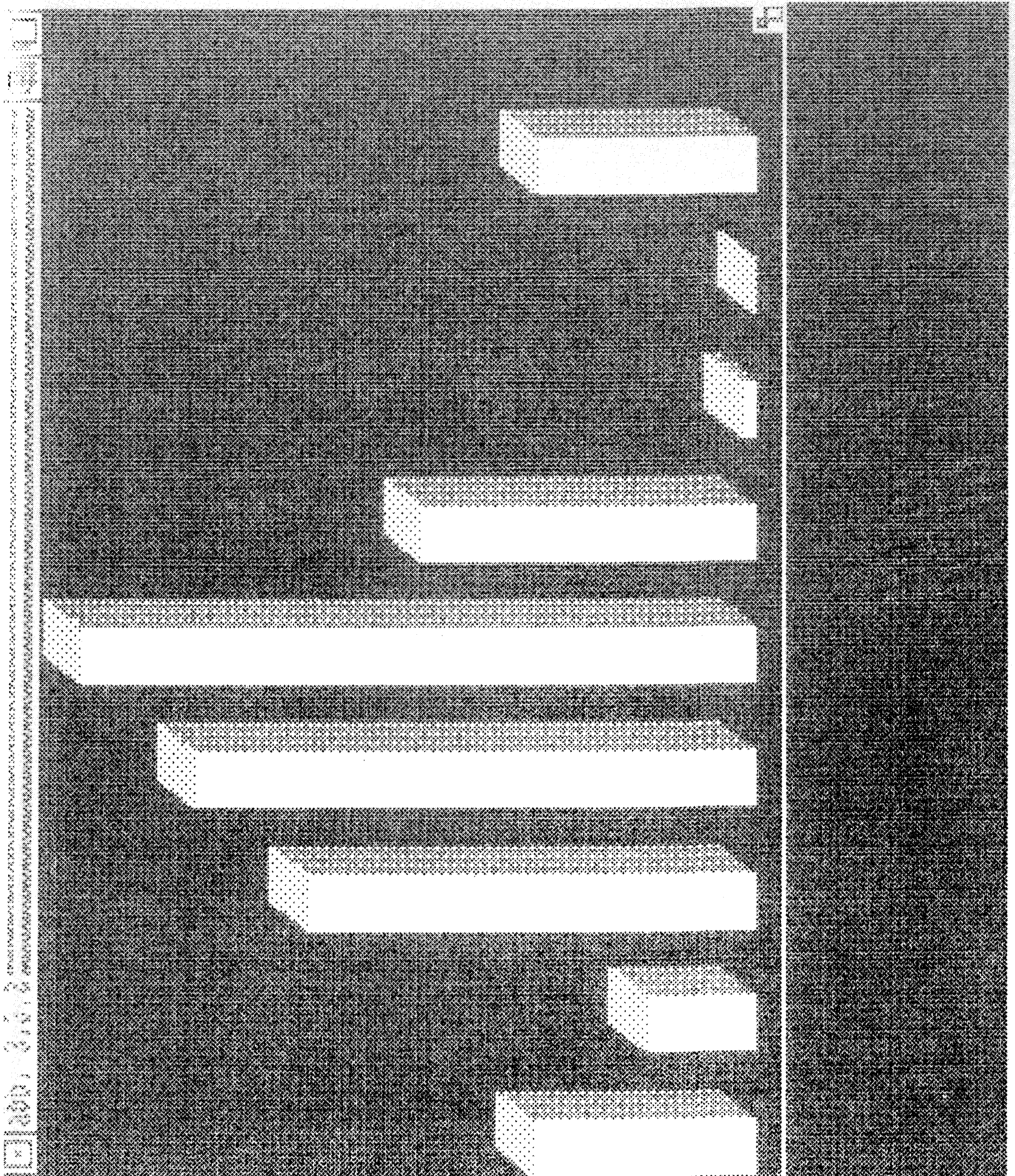


Abb. 3.3.3 Ausgabe des 3D-Säulenprogramms

Aufgaben:
1. Erweitern Sie das Programm, so daß die eingebenen Werte unter den Säulen ausgegeben werden.
2. Fügen Sie die Möglichkeit, Texte neben den Werten einzugeben, hinzu, ähnlich wie bei der Pie-Chart im vorherigen Abschnitt.
3. Programmieren Sie die Ausgabe der Säulen so, daß sich die nächste Säule mit der Seitenfläche der vorhergehenden überdeckt. Dadurch wird der 3D-Effekt noch verstärkt (*Abb. 3.3.4*). Dazu können Sie die Formel für die x-Koordinaten-Berechnung ändern oder die Formel für die Breite. Wenn Sie die Fehlermeldung "Illegal Function Call" bei dem AREA-Befehl erhalten, ist das Gesamtbild zu breit geworden und Sie müssen die Formeln entsprechend korrigieren.

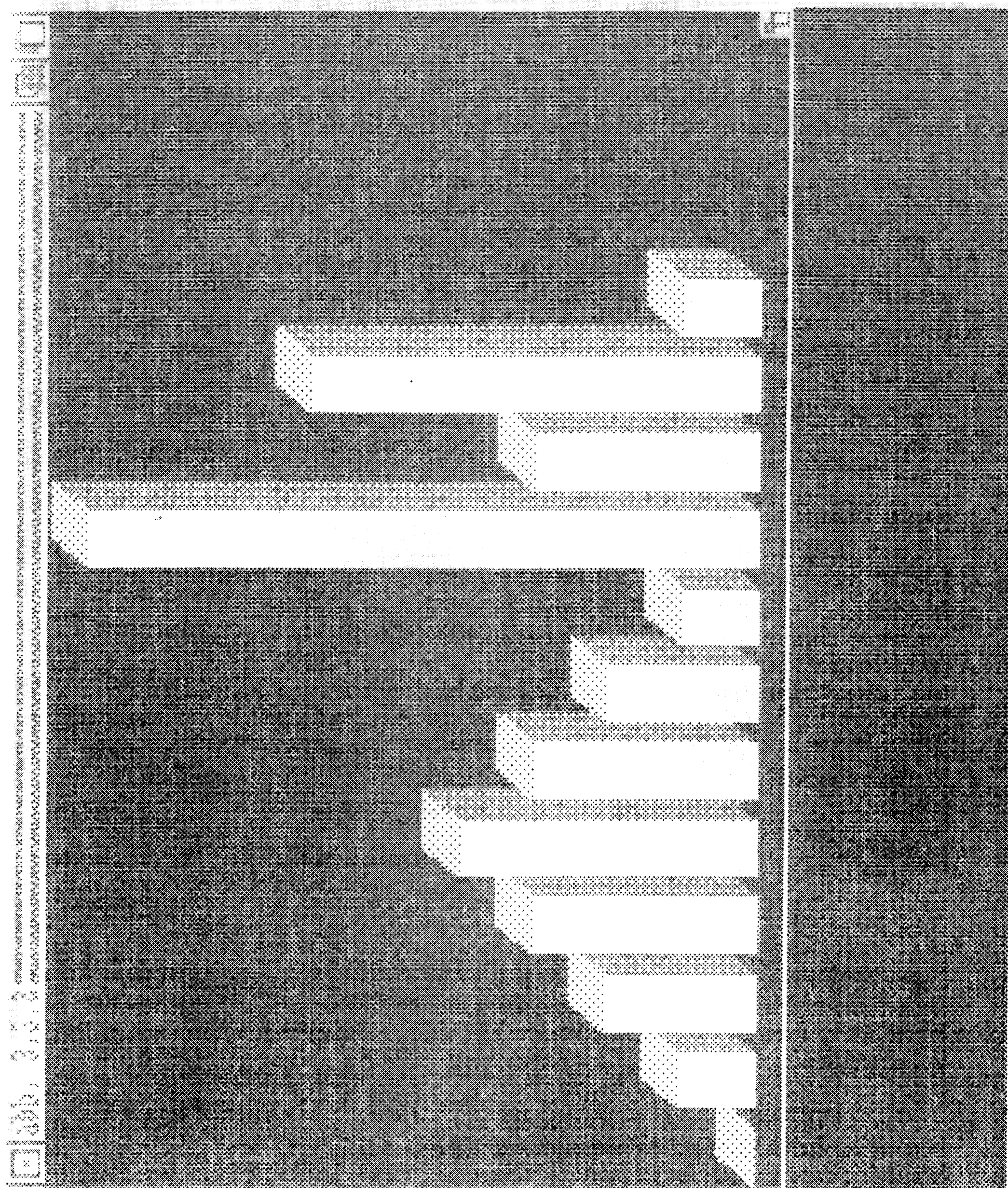


Abb. 3.3.4 3D-Säulen, überlappend

# 4 Menü-Technik

Eine der großen Stärken des Amiga ist die Menü-Technik. Sie haben sicher schon intensiv davon Gebrauch gemacht, zum Beispiel bei der Bedienung des Basic-Interpreters. Dazu gehörig sind natürlich auch die Maus-Befehle, um z.B. die Position der Maus abfragen zu können.
In diesem Kapitel werden Sie die wichtigsten Befehle anhand einiger Beispiele kennenlernen.

**4.1 Auswahl- und Menüleisten**
Menü-Einträge kann man im Amiga-Basic ganz leicht definieren. Dazu gibt es den Befehl MENU. Er besitzt vier Parameter:

```
MENU menükennung,menüpunkt,status,text
```

Die "menükennung" ist eine Zahl zwischen 1 und 10, die angibt, welcher Titel gemeint ist. "menüpunkt" ist eine Zahl im Bereich 0 bis 19. Wenn man den Wert 0 angibt, so wird dadurch der Menütitel definiert, die Zahlen 1 bis 19 geben entsprechende Menüunterpunkte an.
Mit "status" einer Zahl im Bereich 0..2, kann bestimmt werden wie mit dem Menüpunkt zu verfahren ist. Der Wert 0 sagt, daß dieser Menüpunkt gerade inaktiv ist. Ein Wert1 gibt an, daß der Menüpunkt aktiv ist, und der Wert 2 versieht den Menüeintrag mit einer Kennung. Der nachfolgende Text bestimmt den Text in dem Menüfeld. Wenn eine Kennung verwendet wird, muß man darauf achten, daß der Text zwei Leerstellen dafür vorsieht.
Doch genug der Theorie, probieren wir den Befehl einmal aus:

```
MENU 1,0,1,"Test"
MENU 1,1,1,"UnterMenü1"
MENU 1,2,1,"UnterMenü 2"
```

Wenn Sie dieses Programm starten, so passiert scheinbar gar nichts. Auch wenn Sie danach die rechte Maustaste drücken, passiert nichts, denn das Programm hat die Ausführung auch gleich wieder beendet. Wir wollen dies einmal verhindern, indem wir schreiben:

```
WHILE 1=1
WEND
```

und dieses kurze Programmstück hinten anhängen.
Wenn Sie nun das Programm starten, können Sie unsere Menü-Definitionen ganz links oben sehen, sobald Sie die rechte Maustaste drücken. Dann erscheint die Menüleiste:

Test Edit Run Windows

Die drei rechten Einträge stammen vom Basic, um den Befehlsablauf steuern zu können. Wenn Sie den Mauszeiger nun in das Menüfeld "Test" steuern und dabei die

rechte Maustaste gedrückt lassen, so können Sie auch die beiden Texte "UnterMenü 1" und "UnterMenü 2" sichtbar machen.
Wenn Sie nun eines dieser Menüpunkte mit der Maus anfahren und die Taste dort loslassen, also den Menüpunkt selektieren, so passiert nichts weiter. Klar, denn wir haben die Menüpunkt auch noch nicht abgefragt. Stoppen Sie nun das Programm, indem Sie den Menüpunkt "Stop" unter dem Menütitel "Run" selektieren.

Wie kann man die Menüselektion abfragen? Dazu gibt es eine Funktion mit dem Namen MENU(). Diese Funktion besitzt einen Parameter, der 0 oder 1 sein kann. Mit MENU(0) kann man abfragen, welcher Titel ausgewählt wurde. Dies ist die "Menükennung" und mit MENU(1) erhält man dann den "Menüpunkt". Wenn der Wert 0 geliefert wird, so bedeutet dies keine Selektion.
Mit folgendem kleinen Programmstück können wir das ausprobieren:

```
IF MENU(0) <> 0 THEN
 PRINT MENU(1)
END IF
```

Das Programmstück kann man z.B. in die WHILE-WEND-Schleife einbauen und das Testprogramm sieht dann insgesamt so aus:

```
MENU 1,0,1,"Test"
MENU 1,1,1,"UnterMenü1"
MENU 1,2,1,"UnterMenü 2"
WHILE 1=1
 IF MENU(0) <> 0 THEN
  PRINT MENU(1)
 END IF
WEND
```

Probieren Sie das Programm einmal aus. Wenn Sie nach dem Start des Programms das UnterMenü 1 anwählen, so wird die Zahl 1 auf dem Bildschirm ausgegeben. Wenn Sie das UnterMenü 2 anwählen, so wird die Zahl 2 ausgeben.
Wichtig ist noch zu wissen, daß MENU(0) und MENU(1) nach dem Aufruf immer sofort rückgesetzt werden und dann einen Wert 0 als Ergebnis liefern. Will man also das Resultat weiterverwenden, so muß man es vorher in einer Variablen abspeichern. Ein Beispiel dafür sei das folgende Programm:

```
MENU 1,0,1,"Test1"
MENU 1,1,1,"UnterMenü1"
MENU 1,2,1,"UnterMenü 2"
MENU 2,0,1,"Test2"
MENU 2,1,1,"Sub 1"
MENU 2,2,1,"Sub 2"
WHILE 1=1
 title = MENU(0)
 IF title <> 0 THEN
  PRINT title,MENU(1)
 END IF
WEND
```

Nun kann man in der Variablen "title" den Wert von MENU(0) erfahren, also die zuletzt ausgewählte Titelleiste. Übrigens werden Sie jetzt feststellen, daß das Menü

"Edit" fehlt, das zuvor noch da war, da wir dem Basic-Interpreter ein weiteres Menüfeld weggenommen haben.
Sollten Sie Ihr Programm einmal nicht mehr durch das "Run"-Feld stoppen können, da es z.B. durch einen dritten Menütitel überschrieben ist, so können Sie das durch Drücken der Tasten CTRL-C auch direkt erreichen.
Wir wollen den Menüs einmal eine Graphische Ausgabe zuordnen. Man soll einen Kreis oder ein Quadrat auf den Schirm zeichnen können und auf Wunsch soll der Schirm gelöscht werden können.
Das Programm dazu sieht dann so aus:

```
MENU 1,0,1,"Zeichnen"
MENU 1,1,1,"Quadrat"
MENU 1,2,1,"Kreis"
MENU 2,0,1,"Aktion"
MENU 2,1,1,"Loeschen"
WHILE 1=1
 titel = MENU(0)
 IF titel=1 THEN
  auswahl = MENU(1)
  IF auswahl = 1 THEN
   LINE(20,20)-(300,100),1,B
  ELSEIF auswahl = 2 THEN
   CIRCLE(150,50),40
  END IF
 ELSEIF titel = 2 THEN
  CLS
 END IF
WEND
```

Starten Sie das Programm. Es gibt jetzt zwei Menütitel. Wählen Sie unter dem Menütitel "Zeichnen" den Menüpunkt "Quadrat" aus, und es erscheint ein Quadrat auf dem Bildschirm, bei "Kreis" erscheint zusätzlich ein Kreis. Mit dem Menüpunkt "Loeschen" unter dem Menüpunkt "Aktionen" kann man den Schirm auch wieder löschen.
Auf diese Weise kann man schon menügesteuerte Programme schreiben. Doch im Moment arbeitet unser Programm nicht besonders effizient. Dauernd muß abgefragt werden, ob ein Menüpunkt selektiert wurde. Wenn man im Hauptprogramm noch andere Dinge zu tun hat, ist das eigentlich keine vernünftige Lösung. Das Amiga-Basic bietet aber noch eine elegante Möglichkeit, das Programm anders zu gestalten. Dazu gibt es neue Befehle:

```
ON MENU GOSUB unterprogramm
MENU ON
MENU OFF
```

und

```
MENU STOP
```

Was hat es damit auf sich? Der erste Befehl ON MENU GOSUB unterprogramm bewirkt, daß wenn ein Menüpunkt selektiert wird, sofort das Unterprogramm mit dem angegebenen Namen aufgerufen wird. Dabei muß dieses Unterprogramm mit einer Marke anfangen und mit RETURN beendet werden (also nicht SUB ... END SUB verwenden).

Der Befehl ON .. GOSUB .. braucht dabei aber nur einmal im Hauptprogramm ausgeführt werden, um aktiv zu bleiben. Der Basic-Interpreter merkt sich den Befehl und den Namen des Unterprogramms. Nun muß man nicht mehr selbst dauernd abfragen, sondern das übernimmt sozusagen der Basic-Interpreter.
Bevor man diese Fähigkeit verwenden kann, muß man den Befehl MENU ON ausführen. Erst danach können Unterbrechungen des Programms auftreten. Mit MENU OFF kann man die Unterbrechungsfähigkeit auch wieder sperren. MENU STOP sperrt sie auch, jedoch merkt sich der Basic-Interpreter eventuell vorkommende Selektionen und nach einem MENU ON werden sie ausgeführt.
Das Verfahren nennt man übrigens auch Interrupt-Steuerung. Das besondere daran ist, daß ein Programm durch ein Ereignis (hier Selektion eines Menüpunktes) unterbrochen wird, ein Unterprogramm ausgeführt wird und anschließend das unterbrochene Programm wieder weiter ausgeführt wird. Man muß beim Interruptbetrieb darauf achten, daß keine im restlichen Programm gebrauchten Variablen verändert werden und so nach Rückkehr ggf. Störungen des Programmablaufs auftreten können. Dies ist insbesondere beachtenswert, da man ja die Stelle, wo das Programm unterbrochen wird, nicht kennt.
Hier nun ein einfaches Programm, das die neuen Befehle verwendet, *Abb. 4.1.1* zeigt das Listing.
Wenn man das Programm startet, so wird im unteren Bildteil ein durchlaufendes Band von Linien gezeichnet.Dabei werden alle Linien von links nach rechts gezeichnet und danach anschließend wieder gelöscht. Nun kann man jederzeit durch Drücken der rechten Maustaste einen Menüpunkt anwählen und ausführen lassen. Hier läßt sich das Programm übrigens elegant durch Anwahl des Menüpunktes "Quit" unter dem Menütitel "Aktion" abbrechen. Dabei wird der Linienzug im unteren Bildteil noch gelöscht.
Während man die rechte Maustaste drückt, bleibt allerdings das Hauptprogramm stehen, doch dies wird vom Amiga-Betriebssystem so verwaltet und läßt sich nicht verhindern.

```
MENU 1,0,1,"Zeichnen"
MENU 1,1,1,"Quadrat"
MENU 1,2,1,"Kreis"
MENU 2,0,1,"Aktion"
MENU 2,1,1,"Loeschen"
MENU 2,2,1,"Quit"
quit = 0
ON MENU GOSUB unterbrechung
MENU ON
WHILE quit = 0
 FOR x=0 TO 300  STEP 3
  LINE (x,100)-(x,160),3
 NEXT x
 FOR x=0 TO 300  STEP 3
  LINE (x,100)-(x,160),0
 NEXT x
WEND
END
```

```
unterbrechung:
 titel = MENU(0)
 IF titel=1 THEN
   auswahl = MENU(1)
   IF auswahl = 1 THEN
     LINE(20,20)-(300,100),1,B
     MENU 1,1,0
   ELSEIF auswahl = 2 THEN
     CIRCLE(150,50),40
     MENU 1,2,0
   END IF
 ELSEIF titel = 2 THEN
   auswahl = MENU(1)
   IF auswahl = 1 THEN
    CLS
    MENU 1,1,1
    MENU 1,2,1
   ELSEIF auswahl = 2 THEN
    quit = 1
   END IF
 END IF
RETURN
```

Abb. 4.1.1
Menü-Technik

Nun noch zu ein paar besonderen Feinheiten. Bisher haben wir als Status beim Befehl MENU immer den Wert 1 angegeben. Interessant sind aber auch die beiden anderen Werte. Wenn man ein MENU mit dem Wert 0 als Status aufruft, so wird der Eintrag auf dem Bildschirm gepunktet dargestellt (ghosted). Man sieht den Menüeintrag zwar, kann ihn jedoch nicht anwählen. Wenn man den Status eines Menü-Punktes verändern will, so muß man allerdings den Text weglassen, sonst gibt es Probleme. *Abb. 4.1.2* zeigt ein einfaches Beispiel. Dabei wird immer der angewählte Menüpunkt "Quadrat" bzw. "Kreis" gesperrt, sobald er ausgeführt

```
MENU 1,0,1,"Zeichnen"
MENU 1,1,1,"Quadrat"
MENU 1,2,1,"Kreis"
MENU 2,0,1,"Aktion"
MENU 2,1,1,"Loeschen"
MENU 2,2,1,"Quit"
quit = 0
ON MENU GOSUB unterbrechung
MENU ON
WHILE quit = 0
 FOR x=0 TO 300  STEP 3
  LINE (x,100)-(x,160),3
 NEXT x
 FOR x=0 TO 300  STEP 3
  LINE (x,100)-(x,160),0
 NEXT x
WEND
END
```

```
unterbrechung:
 titel = MENU(0)
 IF titel=1 THEN
   auswahl = MENU(1)
   IF auswahl = 1 THEN
     LINE(20,20)-(300,100),1,B
   ELSEIF auswahl = 2 THEN
     CIRCLE(150,50),40
   END IF
 ELSEIF titel = 2 THEN
   auswahl = MENU(1)
   IF auswahl = 1 THEN
    CLS
   ELSEIF auswahl = 2 THEN
    quit = 1
   END IF
 END IF
RETURN
```

Abb. 4.1.2
Menüs mit Ghosted Einträgen

wurde. So könnte man dem Benutzer z.B. andeuten, daß nach einmaligem Zeichnen des Quadrats es nicht besonders sinnvoll ist, erneut ein Quadrat darüber zu zeichnen. Wenn man "Loeschen" aufruft, werden beide Menüpunkte wieder aktiviert, denn dann ist der Bildschirm ja wieder frei.
Nun fehlt uns noch der Status mit dem Code 2. Dann wird ein kleiner Haken vor den Text gesetzt, der dem Benutzer anzeigen soll, daß der betreffende Menüpunkt aktiv ist. Dazu muß man aber bei den Texten zwei Zeichen leer lassen.
Man verwendet diesen Mode, um eine Funktion ein- oder auszuschalten, so z.B. die Zeichenfarbe. *Abb. 4.1.3* zeigt das neue Programm. Wenn man den Punkt "Weiss"

```
MENU 1,0,1,"Zeichnen"
MENU 1,1,1,"Quadrat"
MENU 1,2,1,"Kreis"
MENU 1,3,1,"  Weiss"
MENU 2,0,1,"Aktion"
MENU 2,1,1,"Loeschen"
MENU 2,2,1,"Quit"
farbe = 3
quit = 0
ON MENU GOSUB unterbrechung
MENU ON
WHILE quit = 0
 FOR x=0 TO 300  STEP 3
  LINE (x,100)-(x,160),farbe
 NEXT x
 FOR x=0 TO 300  STEP 3
  LINE (x,100)-(x,160),0
 NEXT x
WEND
END
```

```
unterbrechung:
 titel = MENU(0)
 IF titel=1 THEN
   auswahl = MENU(1)
   IF auswahl = 1 THEN
     LINE(20,20)-(300,100),1,B
     MENU 1,1,0
   ELSEIF auswahl = 2 THEN
     CIRCLE(150,50),40
     MENU 1,2,0
   ELSEIF auswahl = 3 THEN
     IF farbe=3 THEN
      farbe = 1
      MENU 1,3,2
     ELSE
      farbe = 3
      MENU 1,3,1
     END IF
   END IF
 ELSEIF titel = 2 THEN
   auswahl = MENU(1)
   IF auswahl = 1 THEN
    CLS
    MENU 1,1,1
    MENU 1,2,1
   ELSEIF auswahl = 2 THEN
    quit = 1
   END IF
 END IF
RETURN
```

Abb. 4.1.3
Menüs mit Kennzeichnung

anwählt, so werden die Linien des Hauptprogramms ab sofort in der weißen Farbe gezeichnet. Der Menüpunkt ist nun abgehakt und der Haken ist beim erneuten Aufruf sichtbar. Nun kann man den Menüpunkt "Weiss" wieder anwählen, diesmal wird der Haken wieder entfernt und die Linien mit der alten Farbe (Rot) gezeichnet. So verwendet man den Status 2 immer dann, wenn man eine Umschaltfunktion hat: etwas einschalten -- wieder ausschalten. Der eingeschaltete Zustand ist für den Benutzer direkt erkennbar.

## 4.2 Die Maus - Zeichnen mit der Maus

Maus-Befehle hatten wir kurz schon mal in anderen Programmen verwendet, hier wollen wir uns voll und ganz der Maus widmen.
Zur Abfrage der Maus gibt es eine Funktion mit dem Namen MOUSE(). Ferner gibt es auch hier wieder Unterbrechungs-Befehle, auf die wir aber später noch zurückkommen.
Die Funktion MOUSE() besitzt einen Parameter, der für verschiedene Abfragen gedacht ist.

Wenn man MOUSE(0) abfragt, erhält man den Status der linken Maustaste. Die rechte Maustaste kann nicht abgefragt werden, da sie für die Steuerung der Menüs vom Betriebssystem gebraucht wird.
MOUSE(0) liefert den Wert 0, wenn die Maustaste gerade nicht gedrückt ist und auch zwischenzeitlich seit dem letzten MOUSE(0)-Aufruf nicht gedrückt wurde. Nach dem MOUSE(0)-Aufruf werden übrigens die aktuellen Koordinaten des Mauszeigers gespeichert und sind mit anderen Parameterwerten abfragbar.
Liefert der Aufruf eine 1, so ist die linke Maustaste zur Zeit nicht gedrückt, wurde jedoch seit dem letzten Maus-Aufruf einmal gedrückt. Liefert der Aufruf eine 2, so wurde die Maustaste seit dem letzten Aufruf zweimal gedrückt, ist jedoch zur Zeit der Abfrage nicht gedrückt. Der Wert ist einfach ein Zähler, der angibt, wie oft die Maustaste zwischenzeitlich gedrückt wurde.
Wird ein negatives Ergebnis geliefert, so drückt der Anwender die Maustaste zur Zeit noch nieder. Der Wert -1 bedeutet demnach: Maustaste einmal gedrückt und immer noch nicht losgelassen. -2 bedeutet, einmal gedrückt, losgelassen, dann wieder gedrückt und immer noch nicht losgelassen.
Nun muß man natürlich auch Informationen über die Koordinaten bekommen. MOUSE(1) liefert daher die x-Koordinate, die der Mauszeiger beim letzten MOUSE(0)-Aufruf hatte und das unabhängig von der Maustaste. MOUSE(2) liefert entsprechend die y-Koordinate.
Jetzt wird es aber raffiniert.
MOUSE(3) liefert die x-Koordinate, die der Mauszeiger beim Niederdrücken der linken Maustaste hatte, bevor MOUSE(0) aufgerufen wurde, MOUSE(4) liefert die dazugehörige y-Koordinate.
MOUSE(5) verhält sich unterschiedlich. Wenn beim letzten MOUSE(0)-Aufruf die linke Maustaste gedrückt war, so erhält man die x-Koordinate zur Zeit des Aufrufs. Sonst liefert der Aufruf die x-Koordinate, vor dem letzten Aufruf der MOUSE(0), die der Mauszeiger beim Loslassen der linken Maustaste hatte. MOUSE(6) liefert entsprechend die dazugehörige y-Koordinate.
Beginnen wir mit einer einfachen Aufgabe. Wir wollen mit Hilfe der Maus zeichnen. Wenn die linke Maustaste gedrückt wird, so soll der Zeichenvorgang beginnen und solange fortgesetzt werden, bis die Taste wieder losgelassen wird. Dazu muß man prüfen ob die Maustaste gerade gedrückt wird, wenn ja, dann wird mit PSET ein Punkt an der Stelle gezeichnet. Das Ganze geschieht in einer Endlosschleife:

```
WHILE 1=1
 IF MOUSE(0)<0 THEN
  PSET (MOUSE(1),MOUSE(2))
 END IF
WEND
```

Dieses ganz einfache Malprogramm hat noch einige Nachteile. Wenn man die Maus beim Zeichnen schnell bewegt, so bleibt nur eine gepunktete Linie als Ergebnis. *Abb. 4.2.1* zeigt ein Beispiel eines damit erstellten Bildes. Man kann diesen Punkteeffekt natürlich ausnutzen, um Schattierungen zu realsieren.
Dieser Punkteffekt kommt daher, daß wir nur einfache Punkte setzen. Nun wird aber die Maus nur zu bestimmten Zeitpunkten abgefragt, da der Basic-Interpreter auch Zeit für die Ausführung der einzelnen Befehle braucht. Wenn man die Maus

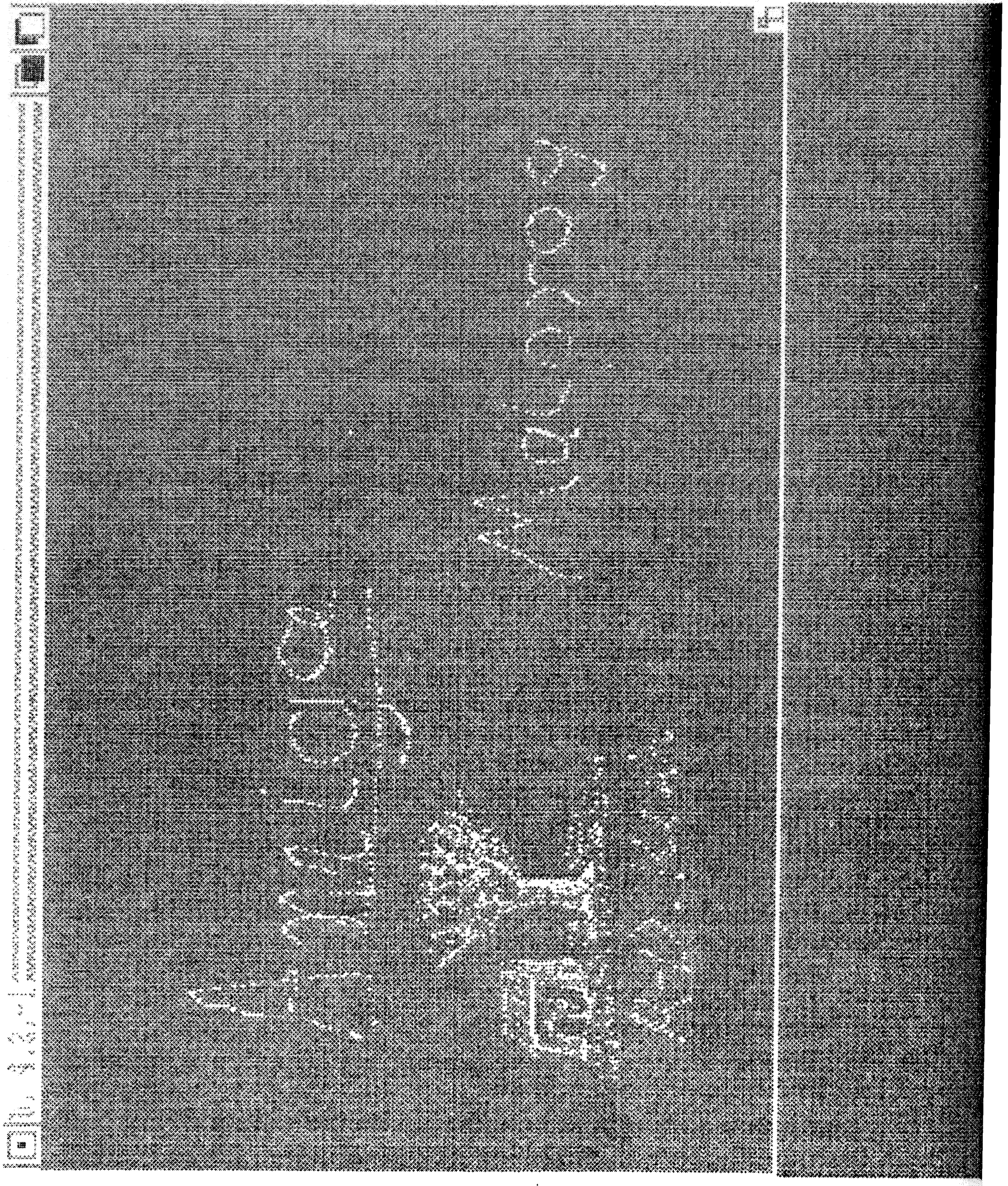


Abb. 4.2.1 Beispiel mit Malprogramm

schnell bewegt, so werden nicht alle Punkte der Strecke erfaßt. Hier hilft ein einfacher Trick weiter. Man zeichnet die Verbindungslinie zwischen alter und neuer Position. Dadurch entsteht dann in jedem Fall eine kontinuierliche Linie. Das neue Programm sieht dann so aus:

```
WHILE 1=1
 IF MOUSE(0) < 0 THEN
  x2 = MOUSE(1)
  y2 = MOUSE(2)
  LINE (x1,y1)-(x2,y2)
  x1 = x2
  y1 = y2
 ELSE
  x1 = MOUSE(1)
  y1 = MOUSE(2)
 END IF
WEND
```

Wenn man nun zeichnet, so entstehen keine gepunkteten Linien mehr. Das Programm arbeitet so: Zunächst ist die Maustaste nicht gedrückt, also wird der ELSE-Teil ausgeführt. Dort wird die aktuelle Koordinate der Maus in x1,y1 gespeichert. Wenn nun die Maus gedrückt wird, so wird MOUSE(0) kleiner Null. Damit wird in den THEN-Teil verzweigt. Dort wird die Koordinate x2,y2 belegt. Nun wird die Linie zwischen x1,y1 und x2,y2 gezogen. Also die Linie zwischen dem letzten Punkt, bei dem die Maus noch nicht gedrückt war und dem ersten Punkt bei dem Sie gedrückt ist. Danach wird in x1,y1 die Endposition gemerkt. Läßt man die Maustaste weiter gedrückt, so wird die Linie von der alten Endposition zur nächsten neuen Position verbunden. Damit erhält man eine fortlaufendene Linie. Unser Programm hat noch einen kleinen Schönheitsfehler. Wenn man die Maustaste während einer Bewegung drückt, so wird als Anfangspunkt der Punkt genommen, bei der die Taste noch nicht gedrückt war. Wenn man das vermeiden will, so muß man als ersten Punkt MOUSE(3),MOUSE(4) verwenden. Die beiden Werte sind aber erst gültig, wenn die Taste schon gedrückt wurde. Man muß die beiden Werte an x1,y1 zuweisen, sobald man in dem THEN-Teil ist und in einem Flag merkt man sich ob der THEN-Teil zum ersten Mal aufgerufen wurde.
Hier das Programm, allerdings ergibt sich beim Lauf kaum ein Unterschied zum vorherigen Programm, denn man kann selbst den genauen Anfangspunkt unter einer Bewegung kaum ausmachen und normalerweise drückt man die Maustaste nicht während man die Maus bewegt, sondern eigentlich immer erst im Stillstand:

```
WHILE 1=1
 IF MOUSE(0)<0 THEN
  IF erstmal = 1 THEN
   erstmal = 0
   x1 = MOUSE(3)
   y1 = MOUSE(4)
  END IF
  x2 = MOUSE(1)
  y2 = MOUSE(2)
  LINE (x1,y1)-(x2,y2)
```

```
  x1 = x2
  y1 = y2
 ELSE
  erstmal = 1
 END IF
WEND
```

Laßt uns mal ein kleines Malprogramm schreiben. Dazu brauchen wir erst mal ein Pflichtenheft.
Was soll das Programm alles können:
1. Malen im Punktemode
2. Malen im Linienmode, also verbundene Linien.
3. Wählen der Farbe (Hintergrund, Color 1,2 und 3)
4. Löschen des Bildschirms mit aktueller Farbe.
Die Auswahl der verschiedenen Modi soll per Menü passieren. Dazu brauchen wir z.B. zwei Menütitel: "Zeichenmode" und "Aktion". Unter "Zeichenmode" soll stehen "Punkte" und "Durchgehend". Unter "Aktion" soll stehen "Farbe 0", "Farbe 1", Farbe 2" und "Farbe 3", sowie "Loeschen" und "Quit".
Die aktuelle Farbe speichern wir in der Variablen "farbe", den Zeichenmode in "zeichenmode". Die Menüsteuerung erfolgt durch Interrupt und in der Hauptschleife fragen wir nur die einzelnen Modi und die Maus ab.
Beginnen wir mit dem Hauptprogramm. Es sieht grob so aus:

```
Menüs initialisieren
Unterbrechungsroutine festlegen
Variablen vorbelegen
Wiederhole bis Quit
  Wenn Maustaste gedrückt
    dann Linien oder Punkte zeichnen je nach Mode
  Sonst
    ggf. rücksetzen von Merkern
  Ende Wenn
Ende Wiederhole.
```

Wie man Linien oder Punkte zeichnet, haben wir schon vorher gehabt, damit kann man diesen Programmteil schon gestalten.
*Abb. 4.2.2* zeigt das Listing. Soweit kann man es auch schon testen.

```
REM kleines Malprogramm
REM Rolf-Dieter Klein
MENU 1,0,1,"Zeichenmode"
MENU 1,1,1,"Punkte"
MENU 1,2,1,"Durchgehend"
MENU 2,0,1,"Aktion"
MENU 2,1,1,"Farbe 0"
MENU 2,2,1,"Farbe 1"
MENU 2,3,1,"Farbe 2"
MENU 2,4,1,"Farbe 3"
MENU 2,5,1,"Loeschen"
MENU 2,6,1,"Quit"
ON MENU GOSUB unterbrechen
```

```
zeichenfarbe = 1
mode = 1 : REM 1=Punkte, 2=Durchg.
quit = 0
MENU ON
WHILE quit = 0
 IF MOUSE(0)<0 THEN
  IF mode = 1 THEN
   PSET (MOUSE(1),MOUSE(2)),zeichenfarbe
  ELSE
   IF erstmal=1 THEN
    erstmal = 0
    x1 = MOUSE(3)
    y1 = MOUSE(4)
   END IF
   x2 = MOUSE(1)
   y2 = MOUSE(2)
   LINE (x1,y1)-(x2,y2),zeichenfarbe
   x1 = x2
   y1 = y2
  END IF
 ELSE
  erstmal = 1
 END IF
WEND
END

unterbrechen:


RETURN
```

Abb. 4.2.2
Das kleine Malprogramm 1. Phase

Nun geht alles um die Unterbrechung. In diesem Programmabschnitt müssen wir zunächst unterscheiden, welches Menü ausgewählt wurde. Beginnen wir also damit:

```
unterbrechen:
 titel = MENU(0)
 WENN titel = 1 DANN
   bearbeiten:  Mode einstellen
 SONST WENN titel = 2 DANN
    bearbeiten:  Farbe oder Löschen oder Quit
 ENDE WENN
RETURN
```

Wir können es so auch schon ausarbeiten:

```
unterbrechen:
 titel = MENU(0)
 IF titel = 1 THEN
  menupunkt = MENU(1)
  mode = menupunkt
 ELSEIF titel = 2 THEN
  Menüpunkt = MENU(1)
```

```
IF Menüpunkt = 5 THEN
  COLOR 1,zeichenfarbe
  CLS
ELSEIF Menüpunkt = 6 THEN
  quit = 1
ELSE
  zeichenfarbe = Menüpunkt-1
END IF
END IF
RETURN
```

Damit kann man schon arbeiten. Wir wollen das Programm aber noch verbessern. Modes die aktiv sind, sollen mit einem kleinen Haken versehen werden, als da sind: "Punkte" und "Durchgehende", sowie die Auswahl der Farben. Zunächst einmal müssen wir dazu je zwei Leerzeichen bei den MENU-Definitionen einbauen. Ferner sind "Farbe 1" und "Punkte" gleich wirksam, müssen also abgehakt werden und bekommen somit den Status 2 gleich bei der Definition. Dann müssen wir das Unterbrechungsprogramm ändern. Bei titel=1 muß das zuvor wirksame Menü mit dem Status 1 versehen werden und das neue mit Status 2. Entsprechend bei der Auswahl der Farben. Das komplette Programm zeigt *Abb. 4.2.3* .Übrigens kann man falsch gezeichnete Punkte oder Linien einfach löschen, indem man als

```
REM kleines Malprogramm
REM Rolf-Dieter Klein
MENU 1,0,1,"Zeichenmode"
MENU 1,1,2,"  Punkte"
MENU 1,2,1,"  Durchgehend"
MENU 2,0,1,"Aktion"
MENU 2,1,1,"  Farbe 0"
MENU 2,2,2,"  Farbe 1"
MENU 2,3,1,"  Farbe 2"
MENU 2,4,1,"  Farbe 3"
MENU 2,5,1,"Loeschen"
MENU 2,6,1,"Quit"
ON MENU GOSUB unterbrechen
zeichenfarbe = 1
mode = 1 : REM 1=Punkte, 2=Durchg.
quit = 0
MENU ON
WHILE quit = 0
 IF MOUSE(0)<0 THEN
  IF mode = 1 THEN
   PSET (MOUSE(1),MOUSE(2)),zeichenfarbe
  ELSE
   IF erstmal=1 THEN
    erstmal = 0
    x1 = MOUSE(3)
    y1 = MOUSE(4)
   END IF
   x2 = MOUSE(1)
   y2 = MOUSE(2)
```

```
   LINE (x1,y1)-(x2,y2),zeichenfarbe
   x1 = x2
   y1 = y2
  END IF
 ELSE
  erstmal = 1
 END IF
WEND
END

unterbrechen:
 titel = MENU(0)
 IF titel = 1 THEN
   menuepunkt = MENU(1)
   MENU 1,mode,1
   mode = menuepunkt
   MENU 1,mode,2
 ELSEIF titel = 2 THEN
   menuepunkt = MENU(1)
   IF menuepunkt = 5 THEN
    COLOR 1,zeichenfarbe
    CLS
   ELSEIF menuepunkt = 6 THEN
    quit = 1
   ELSE
    MENU 2,zeichenfarbe+1,1
    zeichenfarbe = menuepunkt-1
    MENU 2,zeichenfarbe+1,2
   END IF
 END IF
RETURN
```

Abb. 4.2.3
Das kleine Malprogramm, vollständig

Zeichenfarbe, die Farbe des Untergrunds wählt. Als Untergrund kann man auch eine beliebige Farbe wählen, man muß dazu nur nach der Farbwahl den Menüpunkt "Loeschen" anwählen. Danach darf man allerdings nicht vergessen als Zeichenfarbe eine andere Farbe auszuwählen. In *Abb.4.2.4* ist ein kleines Bild dargestellt, so wie man es mit diesem Programm zeichnen kann.

## 4.3 Schalter und Maus

Die Auswahl durch ein Menü in der Titelleiste ist aber nicht die einzige Möglichkeit, um irgendwelche Vorgänge zu starten. Manchmal ist es ganz praktisch, wenn man Knöpfe und Schalter direkt im Bild dauernd sehen kann und diese nur einfach anklicken muß. Davon wird bei vielen Programmen für den Amiga Gebrauch gemacht, so z.B. beim Deluxe-Paint. Dort sind an der rechten Seite des Bildes alle wichtigen und häufig gebrauchten Funktionen in Form kleiner Kästchen untergebracht. Durch Anklicken eines solchen Kästchens kann man dann

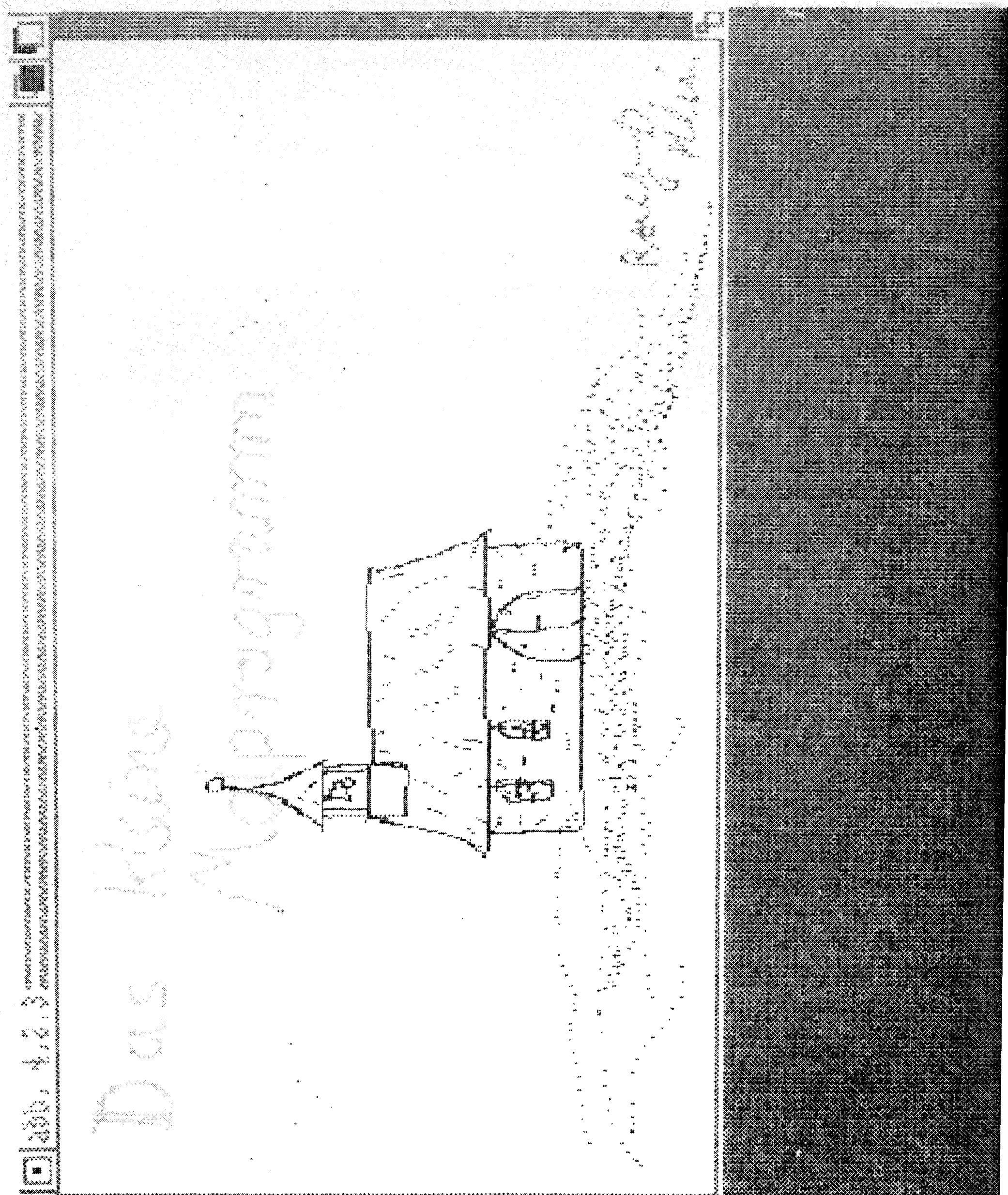



Abb. 4.2.4 Das leistet das kleine Malprogramm

eine Funktion auswählen. *Abb. 4.3.1 (Farbtafel)* zeigt die Menüstruktur des professionellen Malprogramms "Deluxe-Paint-II" von der Firma Electronic Arts. Das gemalte Bild wurde von Electronic Arts als Beispiel mit diesem Programm erstellt. Bei diesem Programm kann man mit den Schaltern (auch Gadgets genannt) verschiedene Zeichenmodi auswählen, die Farbe selektieren oder den Bildschirm löschen. Wie programmiert man solche Schalter? Normalerweise stellt das Amiga-Betriebssystem fertige Funktionen zur Verfügung. Diese sind aber von Basic aus nicht direkt als Befehle verwendbar. Daher programmieren wir uns die Schalter selbst, was auch nicht so kompliziert ist.
Ein solcher Schalter ist ein rechteckiger Kasten. Man muß nun kontrollieren, ob die Maustaste gedrückt ist, und wenn ja, ob sich der Mauszeiger beim Drücken in dem Feld befand. Einen Schalter kann man eindeutig z.B. durch die linke obere Ecke und die rechte untere Ecke definieren, man braucht dann nur abzufragen, ob sich der Mauszeiger in diesem Koordinatenbereich befand. Die *Abb. 4.3.2* zeigt das Schalterfeld mit seinen Koordinaten.
Während man selektiert, sollte der Benutzer einen Hinweis bekommen, daß er sich im Feld befindet. Man kann z.B. das Feld aufleuchten lassen, bzw. den Inhalt invertieren. Man sollte auch die Möglichkeit haben, die Selektion rückgängig zu machen, dadurch daß man z.B. die Maus bei gedrückter Taste wieder aus dem Feld herausbewegt. Das Invertieren des Feldes kann man mit Hilfe des AREAFILL-Befehls realisieren, denn dort kann man den Invertier-Mode angeben.
Zuerst ein kleines Programmstück, das die Aufgabe für ein bestimmtes Feld löst. Dann werden wir das Programm stückweise erweitern.
Zunächst einmal ist es ganz praktisch, wenn wir auch hier wieder die Unterbrechungsfunktion des Amiga-Basic verwenden. Der Befehl lautet: **ON MOUSE GOSUB unterprogramm**. Sobald die linke Maustaste gedrückt wird, verzweigt das Amiga-Basic zum angegebenen Unterprogramm, das mit einem RETURN abgeschlossen wird. Wichtig dabei ist, daß man diesen Befehl nur einmal geben muß und er dann automatisch aktiv bleibt. Zuvor muß man allerding mit **MOUSE ON** die Unterbrechungsfähigkeit noch einschalten.
Das Hauptprogramm sieht dann so aus:

```
ON MOUSE GOSUB tastepruefen
MOUSE ON
```

Nun soll unser Schalter natürlich noch etwas tun, z.B. bei Betätigung den Text "Gedrückt" ausgeben. Ferner darf man nicht vergessen, den Schalter auch grafisch hinzuzeichnen.

```
LINE (300,100)-LINE(340,120),1,B
```

und dann die Hauptschleife:

```
WHILE 1=1
```

das Programm soll endlos laufen.

```
IF schalter = 1 THEN
```

Die Variable "schalter" soll durch das Unterbrechungsprogramm auf 1 gesetzt werden, sobald die Selektion gültig war.

```
PRINT "Schalter gedrueckt"
```

Damit wurde die Meldung ausgegeben.

```
schalter = 0
```

Man darf auch nicht vergessen, die Variable zurückzusetzen und schließlich noch:

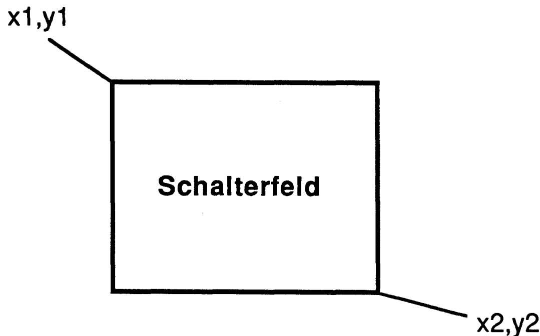



Abb. 4.3.2
Schalterfeld

```
    END IF
WEND
```

Damit steht das Hauptprogramm. Da wir kein SUB..END SUB verwenden, muß man nun noch

```
END
```

hinschreiben, damit auch bei Abbruch der Schleife (1=1 wer weiß wie lange), nicht das Unterbrechungsprogramm angesprungen wird und dort ein RETURN WITHOUT GOSUB-Error erfolgt.
Damit können wir an die Programmierung von "tastepruefen" herangehen.

```
tastepruefen:
```

Nur wenn die Maus beim Drücken im Feld war, kann der Schalter gemeint sein. Dazu programmieren wir:

```
art = MOUSE(0)
```

In Art steht dann die Art der Betätigung, also ob die Maus schon wieder losgelassen wurde, o.ä. Auch kann eine zusätzliche Abfrage nicht schaden:

```
IF art <> 0 THEN
```

Die Werte x und y werden dann erst mal zwischengespeichert, auch um die Basic-Ausdrücke nicht zu lange werden zu lassen:

```
x = MOUSE(1)
y = MOUSE(2)
```

Dann kann man den Bereich abfragen. Achtung, die nachfolgenden Zeilen müssen beim Amiga-Basic in eine einzige Zeile geschrieben werden:

```
IF (x >= 300) AND (x<=340)
  AND (y>=100) AND (y<=120) THEN
```

Nun kann man sicher sein, daß das Feld angesprochen wurde. Wir invertieren daher erst mal das ausgewählte Feld.

```
AREA (300,100)
AREA (340,100)
AREA (340,120)
AREA(300,120)
AREAFILL 1
```

Nun müssen wir prüfen, ob die Maustaste noch gedrückt ist und solange in einer Schleife bleiben, bis entweder die Maustaste losgelassen wird oder das Feld verlassen wurde.

```
WHILE (MOUSE(0)<0 ) AND (x >= 300) AND (x<=340)
 AND (y>=100) AND (y<=120)
 x = MOUSE(1)
 y = MOUSE(2)
WEND
```

Dabei wird in der Schleife nur der x,y-Wert aktualisiert.
Nun können wir erst mal das invertierte Feld wieder zurückschalten:

```
AREA (300,100)
AREA (340,100)
AREA (340,120)
AREA(300,120)
AREAFILL 1
```

Durch den erneuten Aufruf wird der ursprüngliche Inhalt des Feldes wieder hergestellt. Nun muß man noch abfragen, ob die Maustaste nicht mehr gedrückt ist, dann setzt man die Variable "schalter" auf 1, andernfalls wurde nämlich das Feld mit gedrückter Taste verlassen und man ändert dann die Variable "schalter" nicht.

```
IF MOUSE(0)=0 THEN
  schalter = 1
END IF
```

Und schließlich noch:

```
END IF
RETURN
```

*Abb. 4.3.3* zeigt das komplette Programm. Führen Sie nun ein paar Versuche damit aus. Klicken Sie das umrahmte Feld an. Links oben im Bild muß der Text "Schalter gedrueckt" erscheinen. Das Feld muß während des Anklickens orange mit schwarzem Rand aufleuchten. Klicken Sie das Feld erneut an, diesmal ziehen Sie aber den Mauszeiger aus dem Feld, während Sie die Maustaste gedrückt lassen. Das Feld muß sofort wieder in den ursprünglichen Zustand (nur weißer Rand) zurückspringen, sobald der Mauszeiger das Feld verläßt.
Aufgaben:
1. Wenn man das Feld sehr oft anklickt, wird es plötzlich nach oben geschoben, warum? Wie kann man dagegen etwas unternehmen (ohne gleich den Basic-Interpreter neu zu schreiben).
2. Zeichnen Sie auch einmal eine Figur in das Feld. Beim Anklicken darf die Figur nicht zerstört werden. Sie können dieses Programm dazu z.B. mit dem Malprogramm kombinieren. Achten Sie dabei aber auf die neue Situation bei der Mausabfrage, da hier mit einer Unterbrechung gearbeitet wird.

Wir wollen unser Schalterprogramm aber noch verbessern. Zunächst einmal sollen alle Schalter in einem Feld eingetragen sein, so daß man nicht mehr mühsam ein Feld nach dem anderen in einer gesonderten IF-Anweisung abfragen muß.
Auch die Ausgabe der Felder soll durch ein Unterprogramm erfolgen. Allerdings soll die Beschriftung der Felder hier der Einfachheit halber noch durch Einzelbefehle erfolgen.
Wir brauchen also zwei Unterprogramme, das erste Unterprogramm soll felderaus(feld()) und das zweite pruefefeld(feld(),wert) heißen. Das erste Unterprogramm zeichnet alle Feldeinträge und verwendet dazu ein mit READ

```
REM kleines Schalter-Programm
ON MOUSE GOSUB tastepruefen
MOUSE ON
LINE (300,100)-(340,120),1,b
WHILE 1=1
 IF schalter = 1 THEN
  PRINT "Schalter gedrueckt"
  schalter = 0
 END IF
WEND
END

tastepruefen:
 art = MOUSE(0)
 IF art <> 0 THEN
  x = MOUSE(1)
  y = MOUSE(2)
  IF (x>=300) AND (x<=340) AND (y>=100) AND (y<=120) THEN
   AREA (300,100)
   AREA (340,100)
   AREA (340,120)
   AREA (300,120)
   AREAFILL 1
   WHILE (MOUSE(0)<0) AND (x>=300) AND (x<=340) AND (y>=100) AND (y<=12
    x = MOUSE(1)
    y = MOUSE(2)
   WEND
   AREA (300,100)
   AREA (340,100)
   AREA (340,120)
   AREA (300,120)
   AREAFILL 1
   IF MOUSE(0)=0 THEN
    schalter = 1
   END IF
  END IF
 END IF
RETURN
```

Abb. 4.3.3 Kleines Schalterprogramm

zuvor eingelesenes Feld mit folgendem Aufbau:

x1,y1,x2,y2,wert,x1,y1,x2,y2,wert ... -1,-1,-1,-1,-1

Die Zahl -1 soll dabei als Ende-Kennung dienen, so daß man die Anzahl der Feldelemente nicht mühsam abzählen muß.
Prüfefeld wird dann im Unterbrechungsprogramm aufgerufen und fragt dort alle Feldelemente ab. Dabei prüft es die Grenzen und liefert ggf. in der Variablen "wert" das Ergebnis einer Selektion.
Beginnen wir mit dem Unterprogramm "felderaus":

**SUB felderaus(feld()) STATIC**

Durch die Klammerpaare hinter "feld" wird dem Basic mitgeteilt, daß man ein Feld als Parameter übergeben will. In Klammer kann noch eine Dimensionszahl

mitgegeben werden, wenn das Feld mehr als eine Dimension besitzt, was aber bei uns nicht der Fall ist.
Jetzt kann die Ausgabe der Rechtecke, also unserer Schalterfelder, erfolgen:

```
index = 0
x1 = 0
```

Damit kann man die Schleife beginnen lassen:

```
WHILE x1<> -1
```

Und dann die eigentlichen Koordinaten einlesen:

```
x1 = feld(index)
index = index + 1
y1 = feld(index)
index = index + 1
x2 = feld(index)
index = index + 1
y2 = feld(index)
index = index + 2 : REM +2 da Wert nicht gebraucht
```

Falls das eingelesene x1 nicht gerade -1 ist, kann man das Rechteck zeichnen.

```
 IF x1 <> -1 THEN
   LINE (x1,y1)-(x2,y2),1,B
 END IF
WEND
END SUB
```

Damit haben wir unser erstes Unterprogramm komplett. Nun wollen wir das Unterprogramm möglichst bald testen und dazu brauchen wir ein Hauptprogramm. Das Hauptprogramm hat ferner die Aufgabe, aus einem Datenfeld die notwendigen Feldwerte einzulesen:

```
DIM  schalter(100)
```

In diesem Feld sollen alle Werte gespeichert werden. Man kann es mit einem festen Wert belegen, wenn man sicher weiß, daß nicht mehr Werte vorkommen. Eine Angabe von 100 Feldelementen reicht für 100/5 = 20 Einträge inklusive der -1-Kennung für das Ende.

```
i = 0
x1 = 0
WHILE x1 <> -1
  READ schalter(i),schalter(i+1),schalter(i+2)
  READ schalter(i+3),schalter(i+4)
  x1 = schalter(i)
  i = i + 5
WEND
```

mit

```
felderaus schalter()
```

kann man die Schalterfelder dann auf den Bildschirm bringen. Dazu braucht man allerdings noch Definitionen für die READ-Anweisung:

```
DATA 300,100,340,120,1
DATA 250,130,290,150,2
DATA 300,130,340,120,3
DATA 350,130,390,150,4
DATA 300,160,340,180,5
DATA -1,-1,-1,-1,-1
```

Dadurch werden vier Schalter definiert, die kreuzförmig angeordnet sind. Damit wollen wir später ein Bedienpult realisieren.
Doch widmen wir uns erst mal dem Unterbrechungsteil:

```
maustaste:
 pruefefeld schalter(),wert
RETURN
```

Durch die Verwendung eines weiteren Unterprogramms wird das Ganze recht übersichtlich. Man könnte in dem Unterbrechungsteil dann auch noch andere Dinge anschließen. Jetzt wird es aber ernst, das Unterprogramm "pruefefeld" ist fällig.

```
SUB pruefefeld(feld(),wert) STATIC
```

Alle Feldelemente müssen überprüft werden, wenn die Maustaste gedrückt ist:

```
        IF MOUSE(0) <> 0 THEN
          i = 0
          x1 = 0
          WHILE x1 <> -1
..1..
          WEND
         END IF
        END SUB
```

Bei dieser Gelegenheit will ich Sie gleich mit einer praktischen Programmiererfahrung bekanntmachen. Wenn man so stark ineinandergeschachtelte IF-Schleifen verwendet, ist es praktisch, zunächst die gesamte Struktur, also inklusive END IF, WEND hinzuschreiben, ohne den inneren Teil genauer auszufüllen. Damit hat man einen Überblick und kann auch das Einrücken genauer durchführen. Da wir einen guten Editor zur Eingabe der Basic-Programme haben, den man mit den Cursor-Tasten elegant steuern kann, ist das Verfahren leicht durchzuführen. Nun können wir das Innere der Schleife (..1..) weiter ausgestalten:

```
x1 = feld(i)
y1 = feld(i+1)
x2 = feld(i+2)
y2 = feld(i+3)
wert = feld(i+4)
i = i + 5
```

Damit sind alle wichtigen Daten des Feldes eingelesen. Nun wird geprüft, ob die Maus im Inneren des Feldes liegt.

```
        x = MOUSE(1)
        y = MOUSE(2)
        IF (x>=x1) AND (x<=x2) AND (y>=y1) AND (y<=y2) THEN
..2..
        END IF
```

Auch hier wieder der Trick mit dem END IF, das schon vorab eingegeben wird. Wir können jetzt bei Punkt ..2.. weitermachen und die Zeilen ausfüllen, die zutreffen, wenn der Mauszeiger in einem Feld liegt. Nun müssen wir noch das Feld invertieren und dann warten, bis die Maustaste wieder losgelassen wird. Danach kann man das Feld wieder freigeben und entscheiden, ob die Taste noch immer gedrückt war oder nicht, genauso wie bei dem ersten Beispiel im Abschnitt 4.3.

```
AREA (x1,y1)
AREA (x2,y1)
AREA (x2,y2)
AREA (x1,y2)
AREAFILL 1
```

Daraus hätte man auch ein Unterprogramm machen können, da wir das Ganze zweimal brauchen.

```
WHILE (MOUSE(0)<0) AND (x>=x1)
      AND (x<=x2) AND (y>=y1) AND (y<=y2)
  x = MOUSE(1)
  y = MOUSE(2)
WEND
```

Dann wieder invertieren:

```
AREA (x1,y1)
AREA (x2,y1)
AREA (x2,y2)
AREA (x1,y2)
AREAFILL 1
```

und schließlich die Abfrage:

```
IF MOUSE(0) < 0 THEN
  wert = 0
END IF
IF MOUSE(0) = 0 THEN
  x1 = -1
END IF
```

Durch den Trick im ELSE IF-Teil wird die Suche nach weiteren Feldern abgebrochen, sobald eines gefunden wurde und damit "wert" als Ergebnis geliefert. Achtung, durch einen unerfindlichen Fehler im Basic-Interpreter ist es nicht möglich, in dem inneren Teil noch ein ELSE zu verwenden. Daher die doppelte Abfrage.

Damit wäre unser Unterprogramm fertig. Nun zum Hauptprogramm. Als erstes testen wir unsere Unterprogramme. Dazu schreiben wir als restliches Hauptprogramm:

```
ON MOUSE GOSUB maustaste
MOUSE ON
WHILE 1=1
  IF wert>0 THEN
    LOCATE 10,2
    PRINT wert
  END IF
WEND
END
```

*Abb. 4.3.4* zeigt das komplette Programm. Zum Test können Sie nach dem Start des Programms die fünf Felder anklicken. Dann muß der Wert des Feldes links im Bild eingeblendet werden. Wir wollen nun unsere Aufgabe erweitern und etwas sinnvolles mit den Schaltern tun. Ein Laufrollen-Kran soll simuliert werden.

```
REM Schalter-Programm
REM Rolf-Dieter Klein
DIM schalter(100) : REM mit reserve
i = 0
x1 = 0
WHILE x1<>-1
 READ schalter(i),schalter(i+1),schalter(i+2)
 READ schalter(i+3),schalter(i+4)
 x1 = schalter(i)
 i = i + 5
WEND
felderaus schalter()
ON MOUSE GOSUB maustaste
MOUSE ON
WHILE 1=1
 IF wert > 0 THEN
  LOCATE 10,2
  PRINT wert
 END IF
WEND
END



DATA  300,100,340,120,1
DATA  250,130,290,150,2
DATA  300,130,340,150,3
DATA  350,130,390,150,4
DATA  300,160,340,180,5
DATA  -1,-1,-1,-1,-1


SUB felderaus(feld()) STATIC
 index = 0
 x1= 0
 WHILE x1<>-1
  x1 = feld(index)
  index = index + 1
  y1 = feld(index)
  index = index + 1
  x2 = feld(index)
  index = index + 1
  y2 = feld(index)
  index = index + 2: REM +2, da Wert nicht gebraucht
  IF x1<>-1 THEN
   LINE(x1,y1)-(x2,y2),1,b
  END IF
 WEND
END SUB

maustaste:
 pruefefeld schalter(),wert
RETURN
```

```
SUB pruefefeld(feld(),wert) STATIC
 IF MOUSE(0) <> 0 THEN
  i = 0
  x1 = 0
  WHILE x1 <> -1
   x1 = feld(i)
   y1 = feld(i+1)
   x2 = feld(i+2)
   y2 = feld(i+3)
   wert = feld(i+4)
   i = i + 5
   x = MOUSE(1)
   y = MOUSE(2)
   IF (x>=x1) AND (x<=x2) AND (y>=y1) AND (y<=y2) THEN
    AREA(x1,y1)
    AREA(x2,y1)
    AREA(x2,y2)
    AREA(x1,y2)
    AREAFILL 1
    WHILE (MOUSE(0)<0) AND (x>=x1) AND (x<=x2) AND (y>=y1) AND (y<=y2)
     x = MOUSE(1)
     y = MOUSE(2)
    WEND
    AREA(x1,y1)
    AREA(x2,y1)
    AREA(x2,y2)
    AREA(x1,y2)
    AREAFILL 1
    IF MOUSE(0)<0 THEN
     wert = 0
    END IF
    IF MOUSE(0)=0 THEN
     x1 = -1
    END IF
   END IF
  WEND
 END IF
END SUB
```

Abb. 4.3.4 Das Schalterprogramm

Diesen Kran wollen wir mit unseren fünf Steuerfeldern bedienen. *Abb. 4.3.5* zeigt die Belegung der Steuertasten. In *Abb. 4.3.6* ist gezeigt, wie der Kran aussehen soll. Es handelt sich um ein ganz schlichtes Modell, bei dem man die Last in x- und y-Richtung bewegen kann. Wir wollen den Kran mit Hilfe eines Unterprogramms "kran" darstellen. Dazu bekommt das Unterprogramm drei Parameter, die x-Koordinate, die y-Koordinate, und zusätzlich die Farbe, mit der Last und Seil gezeichnet werden sollen. Dies brauchen wir, damit wir den Kran später animieren können.

**SUB kran(x,y,farbe%)**

Das Seil beginnt stets bei einer bestimmten Höhe, wir nehmen einmal y=50. Die Last soll 40x10 Punkte haben.

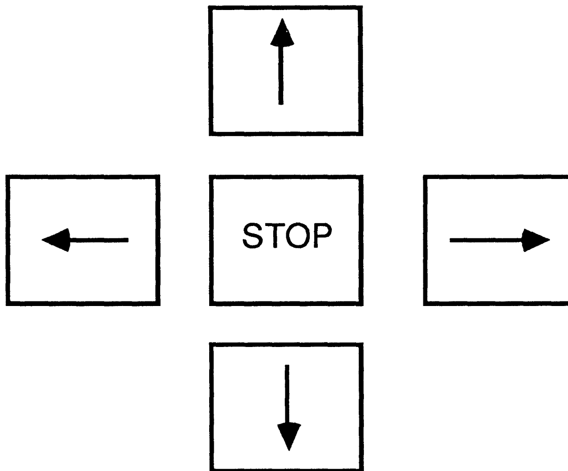



Abb. 4.3.5
Bedienfeld

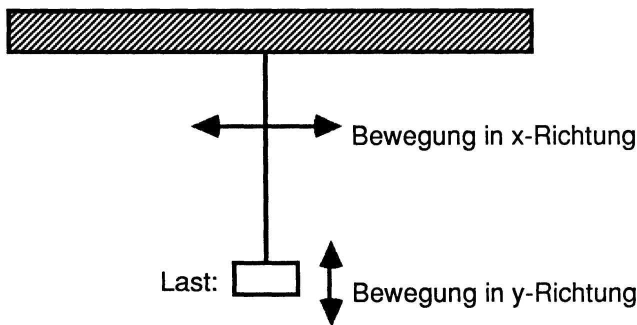



Abb. 4.3.6
Der Kran

**LINE (x,50)-(x,y),farbe%**

zeichnet das Seil und

**LINE (x-20,y)-STEP(40,10),farbe%,BF**
**END SUB**

zeichnet die Last unter das Seil.
Nun zum Hauptprogramm.
Wir können nach MOUSE ON des alten Programms fortfahren. Zunächst brauchen wir noch ein paar Hilfsvariable. Die aktuelle Geschwindigkeit in x- und y-Richtung muß gespeichert werden:

**dirx = 0**
**diry = 0**

Beide Geschwindigkeiten sind zunächst noch null. Dann soll der Kran eine definierte Anfangsposition haben. Ferner brauchen wir noch eine Variable, in der wir uns die alte Position merken.

```
x = 20
xold = 20
y = 60
yold = 60
```

Die alte Position brauchen wir für die Animation. Dabei geht man wie folgt vor: Löschen des Seils mit Last bei der alten Position, zeichnen des Seils mit der Last bei der neuen Position.
Nun zeichnen wir aber zunächst noch die Aufhängung und die Startposition:

```
LINE (18,40)-(222,49),3,BF
kran x,y,1
```

Die Aufhängung wird also in Rot gezeichnet, das Seil mit Last in Weiß.
Nun die Hauptschleife, die lautet:

```
WHILE 1=1
 IF wert > 0 THEN
..1..
 END IF
WEND
```

Wenn Wert nicht größer als null ist, dann wird gar nichts getan, der Kran bleibt stehen. Andernfalls kommen wir in das Innere der Schleife.
Zunächst kann man hier den Wert abfragen und unterschiedliche Aktionen starten.

```
IF wert = 3 THEN
 dirx = 0
 diry = 0
```

Wenn man Stop drückt, werden die Geschwindigkeitsvariablen auf 0 gesetzt.

```
ELSEIF wert = 1 THEN
 diry = -1
```

Die obere Taste belegt diry mit -1, denn die y-Werte des Seils sollen laufend verringert werden.

```
ELSEIF wert = 2 THEN
 dirx = -1
ELSEIF wert = 4 THEN
 dirx = 1
ELSEIF wert = 5 THEN
 diry = 1
END IF
```

So geschieht es entsprechend mit den anderen Werten. Hier wird übrigens nach Abfrage von "wert" dieser nicht mit 0 belegt, sonst würde unsere Schleife nicht mehr durchlaufen und der Kran machte nur einen Schritt. Man kann das aber auch anders lösen, wenn man will.
Man kann nun die Bewegung ausführen:

```
x = x + dirx
y = y + diry
```

Jetzt muß man allerdings sicherstellen, daß Seil und Last im Bildbereich und insbesondere an der Aufhängung bleiben. Man braucht also weitere Abfragen:

```
IF x < 20 THEN x = 20
```

Dies ist eine verkürzte Form der IF THEN END IF-Anweisung. Man spart sich das END IF. Wenn x kleiner ist als 20, dann wird x auf 20 gesetzt. In der Wirkung

bedeutet das, daß der Kran dann stehenbleibt, wenn das Seil an den linken Anschlag kommt.

**IF x > 220 THEN x = 220**

Entsprechend für den rechten Anschlag.
Einziehen kann man das Seil auch nicht über die obere Grenze, also

**IF y<50 THEN y=50**

Achtung, auch hier haben wir es wieder mit der ungewohnten Zählrichtung für die y-Achse zu tun.

**IF y > 180 THEN y = 180**

Denn beliebig viel Seil hat man auch nicht. Nun kann man den Kran bewegen:

**IF (x<>xold) OR (y<>yold) THEN**

..2..

**END IF**

Ist aber nur nötig, wenn auch eine Bewegung erfolgte, also weiter in dem IF-Block ..2..:

**kran xold,yold,0**

Dadurch wird Seil und Last an der vorherigen Stelle gelöscht, denn die Farbe 0 ist auch die Untergrundfarbe.

**kran x,y,1**

Zeichnet das neue Seil und Last in Farbe 1, also Weiß.

**xold = x**
**yold = y**

Damit wird die neue Position zur alten Position und bereitet das nächste Löschen vor.
*Abb. 4.3.7* zeigt das komplette Listing dieses Programms und *Abb. 4.3.8* zeigt eine Kopie vom Bildschirm mit Kran bei der Arbeit.

```
REM Schalter-Programm
REM Rolf-Dieter Klein
DIM schalter(100) : REM mit reserve
i = 0
x1 = 0
WHILE x1<>-1
 READ schalter(i),schalter(i+1),schalter(i+2)
 READ schalter(i+3),schalter(i+4)
 x1 = schalter(i)
 i = i + 5
WEND
felderaus schalter()
ON MOUSE GOSUB maustaste
MOUSE ON
dirx = 0
diry = 0
x = 20
xold = 20
y = 60
yold = 60
LINE (18,40)-(222,49),3,bf
```

```
kran x,y,1
WHILE 1=1
 IF wert > 0 THEN
  IF wert = 3 THEN
   dirx = 0
   diry = 0
  ELSEIF wert = 1 THEN
   diry = -1
  ELSEIF wert = 2 THEN
   dirx = -1
  ELSEIF wert = 4 THEN
   dirx = 1
  ELSEIF wert = 5 THEN
   diry = 1
  END IF
  x = x + dirx
  y = y + diry
  IF x < 20 THEN x = 20
  IF x > 220 THEN x = 220
  IF y < 50 THEN y = 50
  IF y > 180 THEN y = 180
  IF (x <> xold) OR (y <> yold) THEN
   kran xold,yold,0
   kran x,y,1
   xold = x
   yold = y
  END IF
 END IF
WEND
END

SUB kran(x,y,farbe%) STATIC
 LINE (x,50)-(x,y),farbe%
 LINE (x-20,y)-STEP(40,10),farbe%,bf
END SUB



DATA  300,100,340,120,1
DATA  250,130,290,150,2
DATA  300,130,340,150,3
DATA  350,130,390,150,4
DATA  300,160,340,180,5
DATA  -1,-1,-1,-1,-1


SUB felderaus(feld()) STATIC
 index = 0
 x1= 0
 WHILE x1<>-1
  x1 = feld(index)
  index = index + 1
  y1 = feld(index)
```

```
  index = index + 1
  x2 = feld(index)
  index = index + 1
  y2 = feld(index)
  index = index + 2: REM +2, da Wert nicht gebraucht
  IF x1<>-1 THEN
   LINE(x1,y1)-(x2,y2),1,b
  END IF
 WEND
END SUB

maustaste:
 pruefefeld schalter(),wert
RETURN

SUB pruefefeld(feld(),wert) STATIC
 IF MOUSE(0) <> 0 THEN
  i = 0
  x1 = 0
  WHILE x1 <> -1
   x1 = feld(i)
   y1 = feld(i+1)
   x2 = feld(i+2)
   y2 = feld(i+3)
   wert = feld(i+4)
   i = i + 5
   x = MOUSE(1)
   y = MOUSE(2)
   IF (x>=x1) AND (x<=x2) AND (y>=y1) AND (y<=y2) THEN
    AREA(x1,y1)
    AREA(x2,y1)
    AREA(x2,y2)
    AREA(x1,y2)
    AREAFILL 1
    WHILE (MOUSE(0)<0) AND (x>=x1) AND (x<=x2) AND (y>=y1) AN
     x = MOUSE(1)
     y = MOUSE(2)
    WEND
    AREA(x1,y1)
    AREA(x2,y1)
    AREA(x2,y2)
    AREA(x1,y2)
    AREAFILL 1
    IF MOUSE(0)<0 THEN
     wert = 0
    END IF
    IF MOUSE(0)=0 THEN
     x1 = -1
    END IF
   END IF
  WEND
 END IF
END SUB
```

Abb. 4.3.7 Schalterprogramm mit Kran

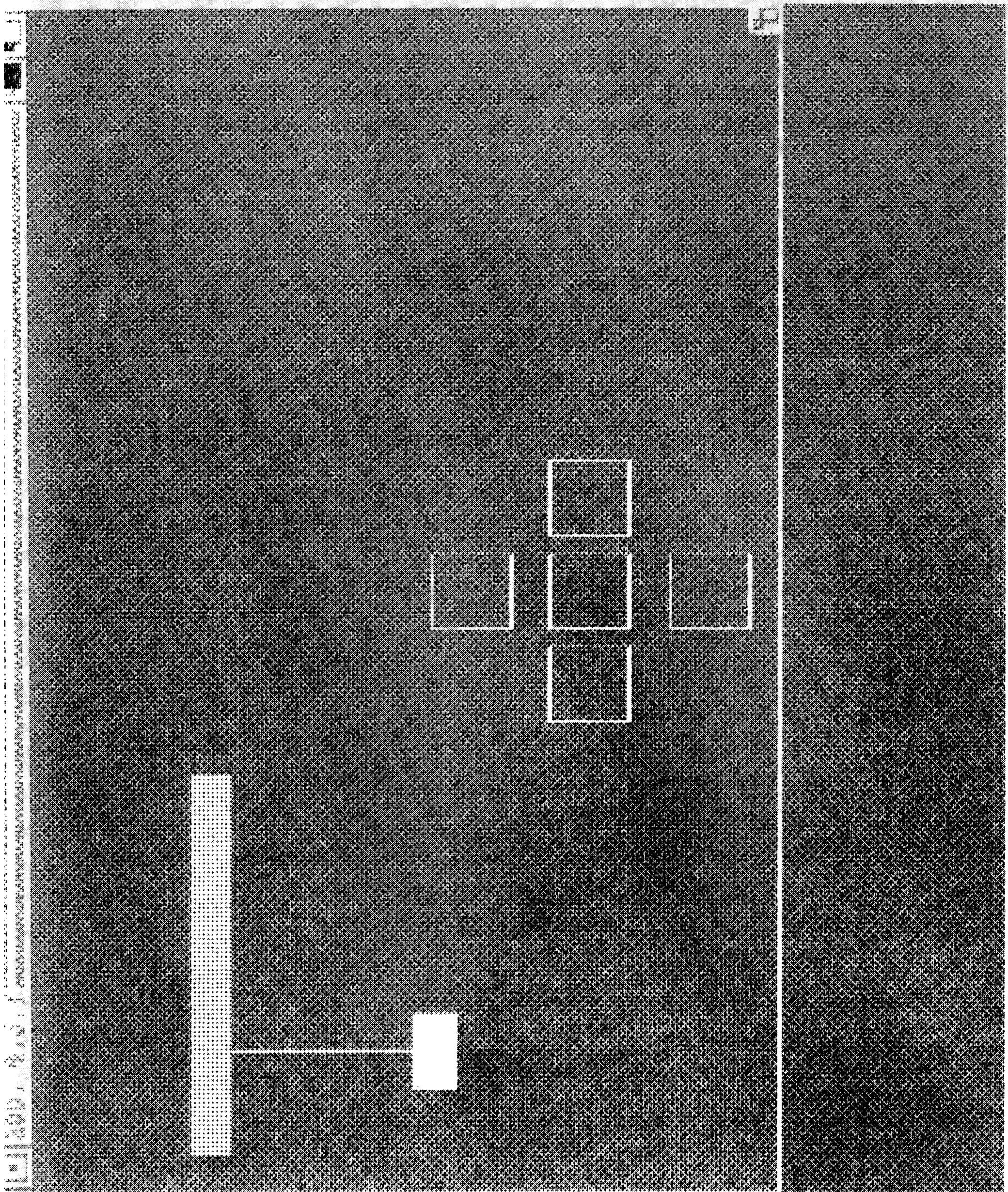


Abb. 4.3.8 Bildschirmkopie des laufenden Krans

Wer will kann dieses Programm noch sehr ausbauen. So sind die Steuertasten noch nicht mit Symbolen ausgefüllt (Pfeile zeichnet man am Besten mit LINE ... usw.). Auch kann man an der Krandarstellung viel tun. Ferner könnte man das Programm zu einem Spiel ausbauen, und unten ein paar Lasten anordnen, die man Heben und irgendwo hinbringen muß. Dazu braucht man dann noch weitere Steuertasten usw. Viel Spaß beim Programmieren.

# 5 Fenster-Technik

Eine der besonderen Erungenschaften der neuen Bedientechnik bei Computern ist die sogenannte Fenster-Technik. Sie kennen die Fenster ja schon von der Bedienung des Basic-Interpreters her, so gibt es dort das Basic-Fenster und das LIST-Fenster. In diesem Kapitel werden wir lernen, wie man selbst Fenster programmieren kann und wie man sogar sogenannte Screens, also große Bildfenster mit mehr Farben, anlegen kann.

## 5.1 Windows - Fenster im Bildschirm

Mit einem neuen Befehl kann man selbst neue Fenster anlegen. *Abb. 5.1.1* zeigt die möglichen Aufrufe des neuen Befehls WINDOW. "kennung" ist dabei eine Zahl größer gleich 1. Sie bezeichnet das Ausgabefenster, so daß man später das Ausgabefenster unter dieser Nummer erreichen kann. Das Basic-Fenster hat dabei die Kennung 1 und man sollte daher für eigene Fenster mit 2 zu zählen anfangen.

```
WINDOW kennung
WINDOW kennung,titelstring
WINDOW kennung,titelstring,(x1,y1)-(x2,y2)
WINDOW kennung,titelstring,(x1,y1)-(x2,y2),typ
WINDOW kennung,titelstring,(x1,y1)-(x2,y2),typ,screen

WINDOW OUTPUT kennung
WINDOW CLOSE kennung
```

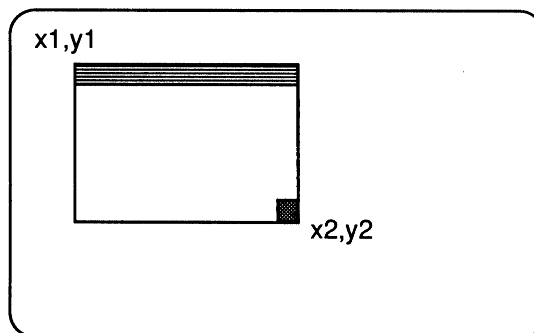



Abb. 5.1.1
Der WINDOW-Befehl

Dieser einfache Befehl legt auch gleich das Fenster an. Wir können also schon ein kleines Testprogramm schreiben, um den Befehl zu testen:

```
WINDOW 2
FOR i=1 TO 10
 PRINT i,SQR(i)
NEXT i
```

Am Schluß sollte man das Fenster wieder schließen, und das geschieht mit WINDOW CLOSE:

```
WINDOW CLOSE 2
```

Nach Start des Programms werden die 10 Berechnungen durchgeführt und dann verschwindet das Fenster wieder. Wenn man den CLOSE-Befehl wegläßt, so bleibt das Ausgabefenster stehen. Man kann es dann auch noch wegbekommen, indem man die linke obere Ecke anklickt. Allerdings sollte man danach auf keinen Fall den Menüpunkt "Show Output" unter dem Menütitel "Windows" anklicken, denn dann kann es zu einem Absturz kommen (Guru-Meditation).
Nun kann man aber beim WINDOW-Befehl noch mehr angeben. Zum Beispiel einen Titel. Wenn Sie die erste Programmzeile des Testprogramms z.B. so abändern können Sie das ausprobieren:

```
WINDOW 2,"Test-Fenster"
```

Interessant wird es nun, wenn Sie zusätzlich die Position und Größe des Fensters angeben. Dabei ist es bei der vorherigen Art übrigens jederzeit möglich, das Fenster zu verkleinern und dann zu verschieben, wenn Sie die Maus zu Hilfe nehmen und die rechte untere Ecke des Fensters greifen, denn die ist für die Größe zuständig und dann, wenn das Fenster kleiner geworden ist, z.B. die Menuleiste anklicken und damit das Fenster verschieben. Unser Programmbeispiel hält jedoch das Fenster nicht lang genug, um diese Manöver auszuführen, Sie müßten dazu z.B. die Ausgabeschleife verlängern.
Wenn man mehr als ein Fenster anlegt, so muß man dem Basic natürlich auch mitteilen können, in welches Fenster die Ausgabe erfolgen soll, dazu gibt es mehrere Möglichkeiten. Mit "WINDOW OUTPUT kennung "kann man die Ausgabe auf das angegebene Fenster umstellen. Durch diesen Befehl bleibt aber die Anordnung der Fenster erhalten, also ein im Hintergrund, bzw. verdecktes Fenster wird nicht wieder sichtbar. Mit "WINDOW kennung" wird das Fenster zusätzlich in den Vordergrund gestellt. Dazu ein paar Beispiele:

```
WINDOW 2,"Fenster1",(10,10)-(300,100)
WINDOW 3,"Fenster 2",(40,80)-(340,160)
FOR i=1 TO 50
  WINDOW OUTPUT 2
  PRINT i,sqr(i)
  WINDOW OUTPUT 3
  PRINT i,sin(i)
NEXT i
WINDOW CLOSE 3
WINDOW CLOSE 2
```

In den beiden erzeugten Fenstern sehen Sie einmal eine Quadratwurzel-Tabelle, zum anderen eine Sinustabelle, quasi gleichzeitig ablaufen. Die beiden Fenster überlappen dabei ein wenig. Löschen Sie einmal die Wörter OUTPUT heraus und

starten das Programm erneut. Nun wird vor jeder Ausgabe das zuvor verdeckte Fenster sichtbar gemacht. Das dauert natürlich viel länger und sieht auch etwas merkwürdig aus. Wenn Sie es schaffen, eines der beiden Fenster zu greifen (anklicken an der Menüleiste) und woanders hinzustellen, so daß die beiden Fenster nicht mehr überlappen, so wird die Ausgabe schneller erfolgen, da nun die jeweils verdeckten Bildteile nicht mehr sichtbar zu machen sind.
Jetzt wird es richtig interessant, wir kommen zu einem weiteren Parameter, den man angeben kann. Der Parameter "typ". Den Wert für Typ erhält man aus einer Addition von verschiedenen zugelassenen Parameterwerten wie folgt:
1 bedeutet, daß die Fenstergröße durch Ziehen des Größensymbols in der rechten unteren Ecke mit Hilfe der Maus verändert werden kann.
2 bedeutet, daß die Position durch Anklicken der Titelleiste mit Hilfe der Maus verändert werden kann.
4 bedeutet, daß das Fenster mit Hilfe des Hintergrund- Vordergrundsymbols durch Anklicken mit der Maus vom Hintergrund in den Vordergrund und umgekehrt gestellt werden kann, d.h. die Verdeckordnung ist änderbar.
8 bedeutet, daß das Fenster durch Anklicken der linken oberen Ecke verschwindet.
16 bedeutet, daß falls das Fenster durch ein anderes Fenster zwischendurch verdeckt war, nach Freigabe des unsichtbaren Teils, die unsichtbaren Inhalte wieder sichtbar werden. Dazu reserviert der Amiga intern einen Extra-Speicher und merkt sich dort die unsichtbaren Bildteile. Er kopiert bei sichtbar werden dieses Teils die unsichbaren Bildteile wieder auf den Bildschirm zurück.
Diese einzelnen Werte kann man nun aufaddieren, um eine Summe von Einzeleigenschaften zu bilden, die für das Fenster gelten sollen. Beispiel:
1+2+16 = 19 bedeutet, daß man das Fenster in seiner Größe ändern kann, daß die Position verändert werden kann und daß unsichtbare Bildinhalte zum Vorschein kommen, wenn sie vorher verdeckt waren.
Als Programmbeispiel nehmen wir den letzten Stand unseres Tests und fügen nur zusätzlich den Typ an, wir wollen dabei die Werte 1 und 2 verwenden und müssen daher als Typ 3 angeben:

```
WINDOW 2,"Fenster 1",(10,10)-(300,100),3
WINDOW 3,"Fenster 2",(40,80)-(340,160),3
FOR i=1 TO 50
  WINDOW 2
  PRINT i,sqr(i)
  WINDOW 3
  PRINT i,sin(i)
NEXT i
WINDOW CLOSE 3
WINDOW CLOSE 2
```

Wenn Sie das Programm so starten, wird Ihnen sofort auffallen, daß jetzt nur eine Zeile der Tabelle in jedem Fenster erscheint. Das kommt daher, daß beim erneuten Anwählen des Fensters durch WINDOW x, der alte Inhalt verlorengeht, denn wir haben ja nicht die Option 16 in den Typ addiert und haben somit keinen Zusatzspeicher reserviert.
Ändern Sie jetzt die WINDOW x-Befehle wieder in WINDOW OUTPUT x um. Die Tabellen kommen dann wieder richtig auf den Bildschirm. Wenn Sie allerdings eines der Fenster verschieben, so bleibt der zuvor verdeckte Teil verloren, denn er

wurde bei diesem Typ nicht gezeichnet und auch nirgends unsichtbar aufbewahrt. Man verwendet also gern die Option 16, so wie sie übrigens auch voreingestellt ist, wenn man keinen Typ angibt (Typ 31, also alle Optionen aktiv, ist voreingestellt), was aber den Nachteil hat, daß der Speicherbedarf ungleich höher ist.
Der letzte Parameter wird für uns erst im nächsten Abschnitt interessant, die Angabe des Screens, also Bildschirms. Screen 1 ist voreingestellt und damit wird die Workbench angesprochen. Als "screen" ist ein Wert zwischen 1 und 4 zugelassen. Man kann also noch drei weitere Bildschirme anlegen, die z.B. unterschiedliche Auflösung besitzen können.
Wir haben aber den WINDOW-Befehl noch nicht ganz abgeschlossen. Es gibt nämlich eine Funktion mit dem gleichen Namen WINDOW(). Sie erhält einen Parameter, mit dem man verschiedene Dinge abfragen kann.
Wenn man WINDOW(0) abfragt, so erhält man die Kennung des gewählten Ausgabefensters. Achtung: Das "gewählte" Ausgabefenster ist nicht unbedingt das Fenster, auf den die Ausgabe erfolgt, es wird durch Anklicken der Maus gewählt, man erkennt es daran, daß die Titelleiste nicht in Geisterschrift erscheint.
Man braucht diesen Befehl, um feststellen zu können in welchem Fenster der Benutzer gerade arbeiten will, z.B. bei einem Grafik-Erstellungsprogramm.
Mit WINDOW(1) erhält man die Kennung des "aktuellen" Ausgabefensters. Es ist dies das Fenster, auf den alle PRINT oder Grafik-Befehle geleitet werden, das also z.B. durch WINDOW OUTPUT x, oder durch WINDOW x aktiviert wurde.
Auch sehr praktisch ist die Funktion WINDOW(2). Man erhält hier die aktuelle Breite des aktuellen Ausgabefensters. Da man die Breite mit Hilfe der Maus ändern kann, wenn das Fenster mit einem Typ definiert wurde, der die Option1 enthält, ist es wichtig, die aktuelle Breite vom Programm aus erfragen zu können. So kann man verhindern, daß Ausgaben in einen Teil des Fensters erfolgen, den der Benutzer gar nicht sieht, weil er das Fenster verkleinert hat. Entsprechend erhält man mit WINDOW(3) die Höhe des aktuellen Ausgabefensters.
WINDOW(4) liefert die x-Koordinate des aktuellen Ausgabefensters, bei der das nächste Zeichen ausgegeben wird und WINDOW(5) liefert die entsprechende y-Koordinate. Mit WINDOW(6) erhält man den höchsten Farbcode, der bei dem aktuellen Ausgabefenster erlaubt ist. Für den Screen 1 ist das also der Wert 3. Sie können das einfach mal ausprobieren, indem Sie PRINT WINDOW(6) angeben.
WINDOW(7) ist schließlich für Fortgeschrittene gedacht, denn damit erhält man eine absolute Speicheradresse des Intuition-Window-Datensatzes, die man braucht, wenn man Maschinenunterprogramm, z.B. direkt vom Amiga verwenden will.
Entsprechend erhält man von WINDOW(8) die Adresse auf den Rasterport-Datensatz. Man braucht die beiden letzten Funktionen aber nur, wenn man mit Maschinenprogrammen arbeitet.
Ein kleines praktisches Beispiel sei hier nicht verschwiegen. Sie haben sicher schon einmal Probleme gehabt, Texte punktgenau zu positionieren. Man kann das nämlich nicht mit dem Amiga-Basic, bzw. wenn dann nur in der x-Richtung (siehe PTAB-Funktion im Amiga-Basic-Handbuch). Hier bekommen Sie ein kleines Programmstück, mit dem es geht. Sie müssen dazu dafür sorgen, daß sich die Datei "graphics.bmap" auf der aktuellen Diskette befindet, denn die enthält alle wichtigen Information für die Ankopplung der Maschinenunterprogramme im Zusammenhang mit dem Grafik-Betrieb. Die Datei befindet sich auf der Original-Basic-Extra-Diskette, wie sie zum Amiga mitgeliefert wird.

Dann schreiben Sie folgendes Programm:

**LIBRARY "graphics.library"**

Damit werden dem Basic die Maschinenunterprogramme des Amiga-Betriebssystems zur Verfügung gestellt. Das neue Unterprogramm soll "text" heißen und drei Parameter besitzen: x,y,textstring. Damit können wir im Hauptprogramm schon mal einen Test schreiben:

**FOR i=1 TO 100**
**x% = i**

Man muß hier eine Integerzahl verwenden.

**y% = i**
**text x%,y%,"Test Hallo"**
**NEXT i**

und schließlich noch

**LIBRARY CLOSE**
**END**

um die nun nicht mehr benötigte Datei "graphics.bmap" freizugeben. Achtung, obwohl man am Anfang des Programms "graphics.library" schreibt wird die Datei "graphics.bmap" angesprochen.

Das Unterprogramm sieht dann so aus:

**SUB text(x%,y%,t$) STATIC**
**CALL Move&(WINDOW(8),x%,y%)**

Das war auch schon der geheimnisvolle Aufruf des Amiga-Betriebssystems. Man achte darauf, daß man hinter Move das Zeichen "&" schreibt. Mit WINDOW(8) wird die Adresse der Rasterport-Datenstruktur an das Maschinenprogramm übergeben.

**PRINT t$**
**END SUB**

*Abb. 5.1.2* zeigt nochmals das gesamte Programm. Sie können das Unterprogramm "text" nun in Ihre eigenen Programme übernehmen und haben damit die Möglichkeit, Texte frei auf dem Bildschirm zu positionieren. Das Testprogramm läßt übrigens den Text von links oben in die linke Mitte des Bildes laufen. Dabei wird durch das dauernde Überschreiben des Textes eine Art Fahne nachgezogen.

```
LIBRARY "graphics.library"

FOR i = 1 TO 100
 x% = i
 y% = i
 text x%,y%,"Test Hallo"
NEXT i

LIBRARY CLOSE
END

SUB text(x%,y%,t$) STATIC
 CALL Move&(WINDOW(8),x%,y%)
 PRINT t$
END SUB
```

Abb. 5.1.2
Das "text"-Unterprogramm für freie Positionierung

Wer Lust bekommen hat, noch mehr mit Amiga-Maschinenprogrammen zu arbeiten, der sei auf den Anhang "Literatur" verwiesen. Man braucht z.B. die vier großen Amiga-Handbücher, um alle Geheimnisse des Amiga zu entdecken.

## 5.2 Screen - Auflösung und Farbe neu definiert

Bisher mußten wir uns mit den vier verschiedenen Farben der Workbench zufriedengeben. Das soll sich jetzt ändern. Wenn man neue Bildschirmformate definieren will, so kann man den Befehl **SCREEN** verwenden.
Der Befehl SCREEN hat folgendes Format:

```
SCREEN n,breite,hoehe,tiefe,modus
```

n ist dabei eine Kennung für den Screen und liegt zwischen 1 und 4. Der Wert 1 ist allerdings schon durch die Workbench belegt und sollte daher nicht verwendet werden.
Der Wert "breite" kann zwischen 1 und 640 liegen. Er gibt die Breite des Bildschirms in Bildpunkten an.
"hoehe" bestimmt die Höhe des Bildschirms und liegt zwischen 1 und 400 Bildpunkten.
"tiefe" ist ein besonderer Wert. Damit wird die Anzahl der Ebenen festgelegt und somit die Anzahl der Farben. Eine kleine Tabelle hilft weiter:

| Tiefe | Anzahl Farben |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |

Es sind also bei Tiefe 5 maximal 32 Farben möglich. Die Farben kann man aus einer Palette von 4096 aussuchen und mit dem PALETTE-Befehl den Farbcodes zuordnen.
Nun gibt es noch den Parameter "modus", er muß zwischen 1 und 4 liegen und bestimmt die Auflösung der Bildschirmdarstellung, ist also auch an "breite" und "hoehe" gekoppelt.
1 bedeutet 320 x 200 Bildpunkte
2 bedeutet 640 x 200 Bildpunkte mit maximal 16 Farben.
3 bedeutet 320 x 400 Bildpunkt mit Zeilensprung
4 bedeutet 640 x 400 Bildpunkt min maximal 16 Farben und Zeilensprung (Achtung, dabei wird sehr viel Speicher belegt).
Mit SCREEN CLOSE n kann man den Bildschirm schließen und damit den belegten Speicher wieder freigeben.
Um auf den Bildschirm zugreifen zu können, muß man auch eine WINDOW-Anweisung verwenden.
Nun ein praktischer Versuch damit:

**SCREEN 2,320,200,5,1**

Damit wird ein Bildschirm mit 320 x 200 Punkten und 32 Farben definiert. Der Mode ist damit 1. Achtung, man kann auch eine Zahl größer 200 eingeben, wenn man über einen PAL-Amiga verfügt, da dort maximal 256 Zeilen zur Verfügung stehen.

```
WINDOW 2,"Neu",,15,2
```

Dadurch wird ein Ausgabefenster definiert. Durch die ",,"-Angabe wird es mit maximaler Größe angelegt. Man könnte es auch durch die Koordinaten definieren, doch da bekommt man ggf. einen "Illegal Function Call", wenn man die Werte zu dicht an die Bildgrenzen legt.
Nun eine kleine Testschleife:

```
FOR i= 0 TO 31
  COLOR i,0
  PRINT "Hallo nun viele Farben"
NEXT i
```

und schließlich nicht vergessen:

```
SCREEN CLOSE 2
```

Wenn man das Programm startet, so erhält man den Text in 32 verschiedenen Farben ausgedruckt. Die Farbpalette ist voreingestellt, man sollte in eigenen Programmen jedoch besser mit dem Befehl PALETTE arbeiten, um auf Nummer sicher zu gehen, daß die Farben auch stimmen. Hier noch ein anderes Testprogramm:

```
SCREEN 2,320,200,5,1
WINDOW 2,"Neu 2",,15,2
FOR i=0 TO 31
  LINE(i*10,20)-(i*10+9,100),i,BF
NEXT i
WHILE MOUSE(0)=0
WEND
SCREEN CLOSE 2
```

*Abb. 5.2.1 (Farbtafel)* zeigt das Ergebnis des Programms. Das Programm kann man beenden, indem man die linke Maustaste an irgendeiner Position drückt.

Aufgaben:
1. Probieren Sie einmal die verschiedenen Auflösungsstufen mit dem SCREEN-Befehl aus.
2. Was passsiert, wenn man für die Breite nicht 320, 640 und für die Höhe andere Werte als 200 und 400 verwenden. Experimentieren Sie damit.

## 5.3 3D-Kugel

An dieser Stelle wollen wir die neuen Erkenntnisse in einem kleinen Programm anwenden, das eine 3-dimensionale Kugel mit Hilfe von Graustufen darstellt. Dazu brauchen wir einen Bildschirm mit 16 Graustufen. übrigens, obwohl der Amiga 32 Farben gleichzeitig auf den Schirm bringen kann, kann man jedoch nicht mehr als 16 verschiedene Graustufen einer Farbe anwählen, da die Anzahl der Stufungen pro Farbkanal begrenzt ist.
Hier will ich noch einen Trick zeigen, mit dem man die Anzahl der Stufungen scheinbar erhöhen kann, das sogenannte Dithering (engl. f. zittern). Das bedeutet, man wechselt zwischen zwei Farben hin und her und erhält so, obwohl mit einer geringeren Auflösung, eine scheinbare Zwischenfarbe.Wir verwenden als Basis die

voreingestellte Farbpalette, da die Farben dort recht gut gewählt sind, allerdings haben wir dann nur 12 Graustufen mit den Farbcodes 20..31. Doch durch die Dither-Technik wird das leicht wieder wettgemacht.
*Abb. 5.3.1* zeigt das fertige Programm. Die Kugel wird durch das Unterprogramm "kugel" erzeugt, die dazu drei Parameter, nämlich die x- und y-Position, sowie den

```
REM 3D-Kugel
SCREEN 2,320,200,5,1
WINDOW 2,"3D-Kugel",,31,2
REM Voreingestellte Palette
REM verwenden, 12 Graustufen.
REM vorbelegung der Dithermatrix
DIM dith(4,4)
FOR y=0 TO 3
 FOR x=0 TO 3
  READ dith(x,y)
 NEXT x
NEXT y
kugel 120!,100!,80!
WHILE 1=1
WEND
SCREEN CLOSE 2
DATA 0,8,2,10
DATA 12 ,4,14,6
DATA 3,11,1,9
DATA 15,7,13,5
END

SUB kugel(x,y,r) STATIC
 FOR x1=-r TO r
  FOR y1=-r TO r
   rquad = x1*x1+y1*y1
   IF rquad < r*r THEN
    farbe = COS(SQR(rquad)/r*3.141592/2)
    dot x+x1,y+y1,farbe
   END IF
  NEXT y1
 NEXT x1
END SUB


SUB dot(x,y,farbe) STATIC
 SHARED dith()
 f% = farbe * 175
 f1% = f% MOD 16
 f2% = f% \ 16
 IF f1% > dith(x MOD 4,y MOD 4) THEN
  f2% = f2% + 1
 END IF
 PSET(x,y),f2%+20
END SUB
```

Abb. 5.3.1
3D-Kugel-Programm

Radius erhält. In dem Unterprogramm wird in einer Schleife für jeden Punkt des umschließenden Quadrats (-r bis +r für x und y) berechnet, ob dieser Punkt auf die Kugel trifft, dann ist nämlich x*x+y*y kleiner oder gleich dem Radius zum Quadrat. Wenn ja, so kann der Farbton berechnet werden. Dazu wird die Cosinus-Funktion verwendet. In der Mitte der Kugel soll die maximale Intensität herrschen, und am Rande soll die Kugel dunkel erscheinen. Die COS-Funktion wird Eins für einen Winkel von 0 Grad und 0 für einen Winkel von 90 Grad. Den Winkel können wir direkt aus dem Radius ermitteln, wenn der abgetastete Punkt genau dem Radius entspricht, dann wird durch die Formel daraus ein Winkel von 90 Grad, bzw. PI/2, da man Winkel in Radiant angeben muß. Wenn der Radius 0 ist, so wird auch der Winkel 0. Nun kann man die so gewonnene Farbe, die im Bereich 0..1 liegt an das nächste Unterprogramm mit dem Namen "dot" geben. Dot besitzt 3 Parameter, x,y und den Farbcode 0..1. Die Koordinate x errechnet sich aus der beim Unterprogramm angegebenen Koordinate x und der Hilfskoordinate x1. Entsprechend bei y.Im Unterprogramm "dot" erfolgt nun das trickreiche dithering. In einem 4 x 4 Feld mit dem Namen "dith" sind Werte zwischen 0 und 15 gespeichert. Doch zunächst einmal wird der Wert von Farbe mit 175 multipliziert. Dieser Wert ergibt sich aus 16 Ditherwerten * (12 Graustufen -1 ) -1. Der Farbecode in f% reicht damit von 0 bis 175. Nun wird der Code für die nicht als Farbe verwertbaren Stufen berechnet. f1% ergibt sich aus der Modulo-Division von f% und 16 und kann damit Werte zwischen 0 und 15 annehmen. In f2% steht der vorläufige Farbcode, der sich aus der Ganzzahl-Division des Farbcodes in f% und 16 ergibt. Also, da in f% ein Wert zwischen 0 und 175 steht ,kann in f2% nur ein Wert zwischen 0 und 10 stehen. Der Wert 11 kommt nicht vor, und wird beim Dithering noch gebraucht.
Nun kommt die Abfrage. Wenn die Farbstufe in f1% einen Wert größer als die Dithermatrix an der Stelle x MOD 4, y MOD 4 hat, dann wird der Wert in f2% um Eins erhöht. x MOD 4 bewirkt, daß nur Werte zwischen 0 und 3 vorkommen und damit die Dithermatrix richtig indiziert wird. Was bedeutet das Ganze? Die Dithermatrix sorgt dafür, daß umsomehr Farbcodes um Eins erhöht werden, je größer der Code in f1% ist, also je näher die Farbe in f% an der nächsten realisierbaren Stufe liegt. Die Dithermatrix ist so belegt, daß das dadurch entstehende Farbmuster nicht sehr stört.
*Abb. 5.3.2* zeigt ein Bildschirmauszug, der mit einem Tintendrucker gemacht wurde. Drucker verwenden normalerweise auch eine Dithertechnik um die Graustufen darzustellen, denn ein Drucker hat pro Farbe normalerweise sogar nur zwei Stufen: Farbe da, Farbe nicht da.
Dennoch sehen die Bilder ganz passabel aus. Den realen Eindruck von dem Programm können Sie aber nur bekommen, wenn Sie die Kugel auf Ihrem Bildschirm sehen. Das Programm macht es Ihnen leicht, weitere Kugeln hinzuzufügen. Sie können als Anregung auch einmal versuchen, andere Körper auf diese Weise darzustellen: Zylinder, Kegel oder Quader. Dann steht einem kleinen 3D-Programm nichts mehr im Wege.

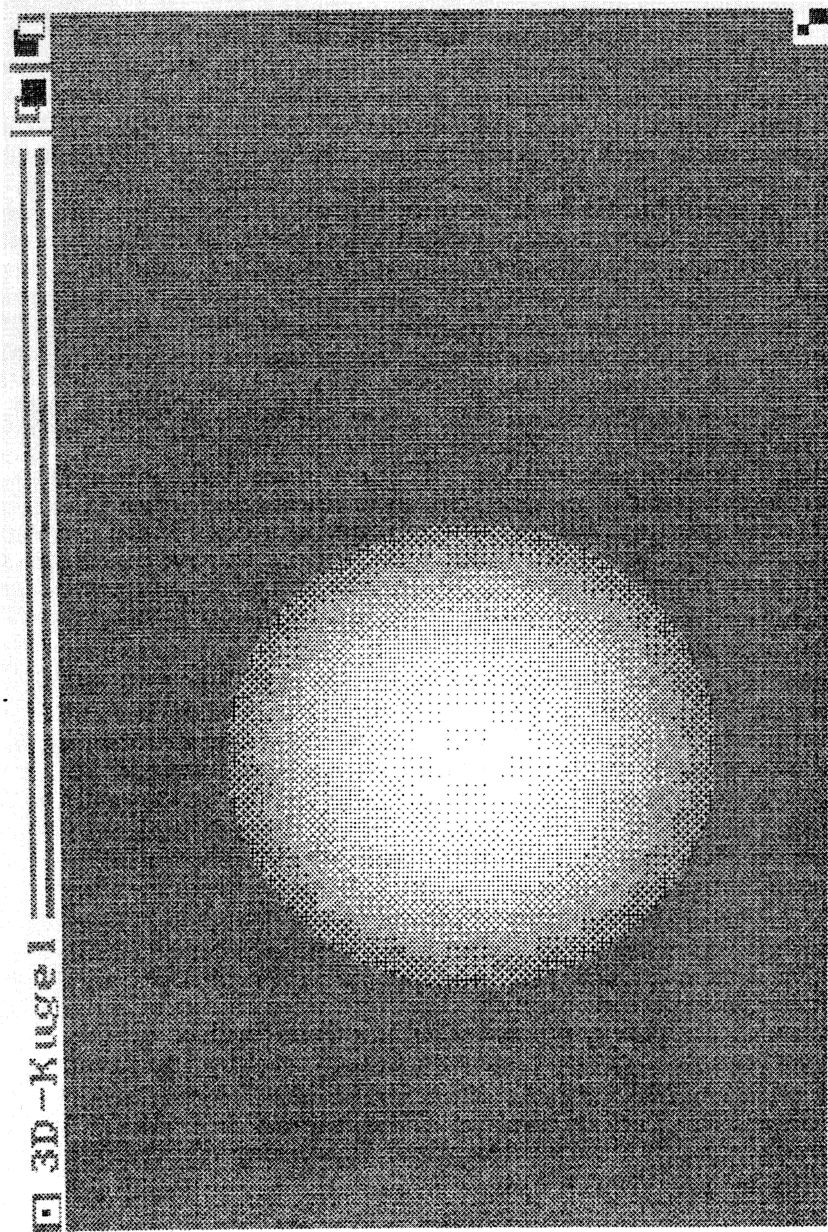



Abb. 5.3.2 Die 3D-Kugel

# 6 Animation

Jetzt bringen wir Bewegung in unsere Bilder. Vielleicht haben Sie schon einige Programme auf dem Amiga gesehen, die animiert sind. Mit unserem Kranprogramm aus Kapitel 4 haben wir ja auch schon einen Schritt in diese Richtung getan. In diesem Kapitel wollen wir systematisch an die Sache herangehen. Es gibt unterschiedliche Techniken der Animation, wobei sich allerdings viele nur beschränkt in der Sprache Basic realisieren lassen.
Das liegt natürlich daran, daß ein Basic-Interpreter immer viel langsamer arbeitet als z.B. eine Sprache wie C oder die Maschinensprache. Allerdings gibt es für die Benutzer des Amigas auch ein Compiler für die Sprache Basic (z.B. AC/BASIC von absoft). Und dann ist auch das Problem der Geschwindigkeit nicht mehr so groß. Ein Basic-Compiler, das nur zur Erinnerung, übersetzt das Basic-Programm in ein Maschinenprogramm. Die Ausführung des Maschinenprogramms ist ungleich schneller als die Interpretation der einzelnen Befehle beim Basic-Interpreter.

## 6.1 Color-Paletten-Animation

Dies ist die einfachste und wirkungsvollste Art der Animation. *Abb. 6.1.1* zeigt ein Schema des Verfahrens. Will man z.B. einen Vorwärtsstrom darstellen, indem sich einzelne Bänder von links nach rechts bewegen sollen, so teilt man den Bändern zunächst einmal unterschiedliche Farbcode zu. Hier z.B. 0,1,2 und 3. Wenn man nun die Bänder in Bewegung versetzen will, so muß man nichts anderes tun, als die den Farbcodes zugeordneten Farben zu ändern. Z.B. ordnet man dem Farbcode 0 in der ersten Animationsphase eine Farbe Weiß zu, den Farbcodes 1, 2 und 3 aber die Farbe schwarz. Ein Band leuchtet dann in Weiß, die anderen sind dunkel und daher auch nicht unterscheidbar. Nach einer kleinen Verweilzeit ordnet man in der zweiten Phase dem Farbcode 1 die Farbe Weiß zu und ändert die Farbe vom Farbcode 0 nach Schwarz. Das Band scheint nach rechts zu wandern. In der dritten Phase ordnet man dem Farbcode 2 die Farbe Weiß zu und dem Farbcode 1 die Farbe Schwarz. Wieder wandert das Band ein Stück nach rechts. In der vierten Phase schließlich verfährt man so mit Code 3 und 2. Das Band wandert wieder ein Stück weiter. Die fünfte Animationsphase schließlich deckt sich wieder mit der ersten Phase. Nun kann man das Ganze wiederholen. Das Band scheint sich fortlaufend nach Rechts zu bewegen.
*Abb. 6.1.2* zeigt ein fertiges Programm, das mit 8 Phasen arbeitet. Zunächst wird im Programm ein neuer Bildschirm mit 32 Farben angelegt, wobei allerdings auch einer mit 16 genügt hätte. Dann wird in einer Schleife das Band ausgegeben. Der Farbcode für jedes Element des Bandes wechselt dabei von 20 bis 31 und dann wieder 20..31 usw. Nun startet die Animation. Dazu werden zwei Variable

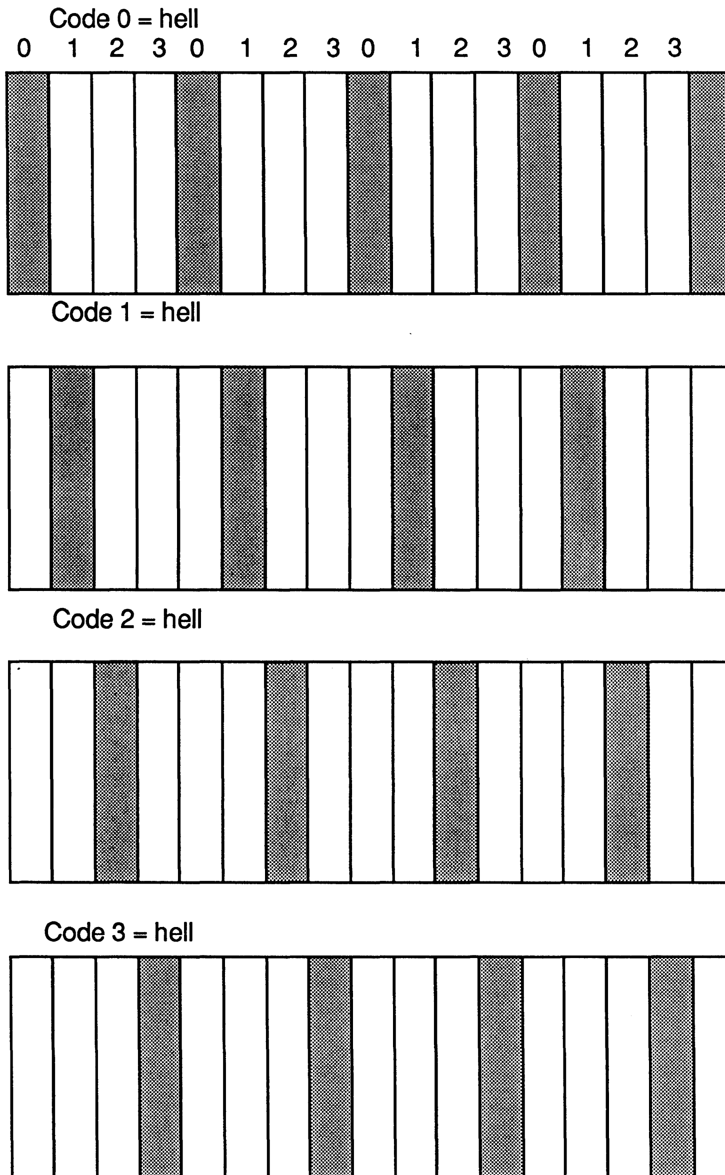



Abb. 6.1.1
Paletten-
Animation

vorbelegt. In "index" merkt sich das Programm die aktuelle Phase und in "oldindex" die Phase, die davor lag. "oldindex" wird mit -1 belegt, damit später das Unterprogramm merkt, daß die Farbpalette noch nicht programmiert wurde. In eine Endlosschleife wird zunächst eine sogenannte Warteschleife ausgeführt, die lautet:

**FOR i=1 TO 100: NEXT i**

```
REM Colorpalettenanimation

SCREEN 2,320,200,5,1
WINDOW 2,"Paletten Animation",,15,2
FOR i=0 TO 100
 x = i*5
 LINE (x,5)-STEP(5,80),(i MOD 12) + 20,bf
NEXT i
index = 0
oldindex = -1
WHILE 1=1
 FOR i=1 TO 100: NEXT i: REM warten
 animate index,oldindex
WEND
WINDOW CLOSE 2
SCREEN CLOSE 2
END

SUB animate(index,oldindex) STATIC
 IF oldindex = -1 THEN
  FOR i=20 TO 31
   PALETTE i,0,0,0
  NEXT i
  oldindex = index
 END IF
 index = index + 1
 IF index > 11 THEN index = 0
 PALETTE index+20,1,1,0
 PALETTE oldindex+20,0,0,0
 oldindex = index
END SUB
```

Abb. 6.1.2
Programm: Paletten-Animation

Die einzige Aufgabe dieses Programmteils ist es, Rechenzeit zu verbrauchen. Es gibt im Amiga auch noch elegantere Möglichkeiten, ein Programm warten zu lassen (siehe Amiga-Handbuch TIMER), doch dieser Weg ist bei vielen Programmieren sehr beliebt. Der Methode hat einen Nachteil. Wenn man das Programm wirklich mal mit einem Compiler übersetzt, so wird auch die Schleife schneller ausgeführt und das kann erheblich sein. Die Warteschleife ist aber bei uns nicht sehr kritisch und dient nur dazu, daß das Band nicht zu schnell von links nach rechts wandert.
Das Unterprogramm "animate" sorgt für die Umschaltung der Farbpalette.
In dem Unterprogramm wird zunächst abgefragt, ob oldindex mit -1 belegt ist, und damit zunächst eine allgemeine Initialisierung ausgeführt werden muß.
Wenn das der Fall ist, werden erst einmal alle Farbcodes im Bereich 20 bis 31 mit der Farbe Schwarz belegt. Dann wird anschließend an "oldindex" der Wert "index" zugewiesen, damit "oldindex" bei weiterer Verwendung einen gültigen Wert hat.
Nun wird der Wert von "index" um Eins erhöht, und damit die nächste Phase angewählt. Anschließend wird noch abgefragt, ob der Wert von "index"größer als 11 ist, denn nur der Bereich 0..11 ist bei uns zulässig, wenn ja, so wird "index" wieder mit 0 belegt und die Phase 0 beginnt erneut.

Schließlich wird mit PALETTE index+20,1,1,0 die Farbe Gelb (Rot=1, Gruen=1 Blau = 0) dem Farbocde index+20 zugeordnet. Danach wird der Farbcode mit oldindex+20 mit Schwarz belegt.
Wenn Sie das Programm starten, bewegt sich ein Band von links nach rechts über den Bildschirm.
Aufgaben:
1. Programmieren Sie ein Band, das von oben nach unten läuft. Hinweis: Man muß nur das Hauptprogramm ändern.
2. Programmieren Sie einen Ring, bei dem ein Band umläuft. Hinweis: Verwenden Sie den Befehl CIRCLE, mit steigendem Radius und entsprechender Winkelangabe für Start und Endewinkel.

Mit dem Paletten-Animations-Verfahren kann man sehr viele Bilder animieren. Das Verfahren läßt sich immer dann anwenden, wenn man eine Bewegung in Phasen zerteilen kann, die sich nach einer bestimmten Zeit wiederholen. Die Anzahl der Phasen darf beim Amiga 32 nicht überschreiten. Bei 32 Phasen bekommt man allerdings Probleme, wenn man noch andere Objekte im Bild haben will, die nicht animiert sind. Für jede nicht animierte Farbe verschwindet eine Phase.

## 6.2 Sprites und Bewegung - Spriteeingabe

Im Amiga-Basic gibt es recht raffinierte Befehle, um Objekte über den Bildschirm bewegen zu können. Dabei werden sogenannte Sprites bzw. Bobs verwendet. Sprites sind kleine Gebilde, die maximal 16 Punkte breit, aber beliebig hoch sein können. Auch können Sprites nur 3 unterschiedliche Farben verwenden. Die Bobs können dageben beliebig groß sein und entsprechend der Bildschirmdefinition Farben verwenden (max. 32). Sie werden jedoch nicht so schnell bewegt wie die Sprites, da die Bewegung durch Kopiervorgänge erfolgt.
Will man Sprites selbst erstellen, so verwendet man das Program "OBJEDIT", wie es mit dem Amiga-Basic mitgeliefert wird. Bitte lesen Sie die Bedienung im Handbuch Amiga-Basic nach, da sie sich ggf. ändern könnte.
Auf der Beispiel-Diskette befindet sich auch schon ein fertiger Sprite, den wir dann einfach verwenden wollen, um die Befehle auszuprobieren. Er trägt den Namen "ball".
Es gibt eine große Zahl von Befehlen, die Sprites unterstützen, am Besten gehen wir dabei Schritt für Schritt vor.
*Abb. 6.2.1* zeigt den wichtigsten Befehl. Damit kann man ein Objekt definieren. Als Definition verwendet man eine Datei, wie sie vom OBJEDIT erstellt wurde. Der Aufruf sieht dann z.B. so aus:

**OPEN "ball" FOR INPUT AS 1**

Damit wird die Datei angemeldet. Nun kann man sie direkt in das Objekt einlesen:

**OBJECT.SHAPE 1,INPUT$(LOF(1),1)**

LOF stellt dabei die Länge der Datei fest und INPUT$ liest diese Daten direkt ein.

```
OBJECT.SHAPE objekt,definition
OBJECT.SHAPE objekt1,objekt2
```

Abb. 6.2.1 OBJECT.SHAPE-Befehl

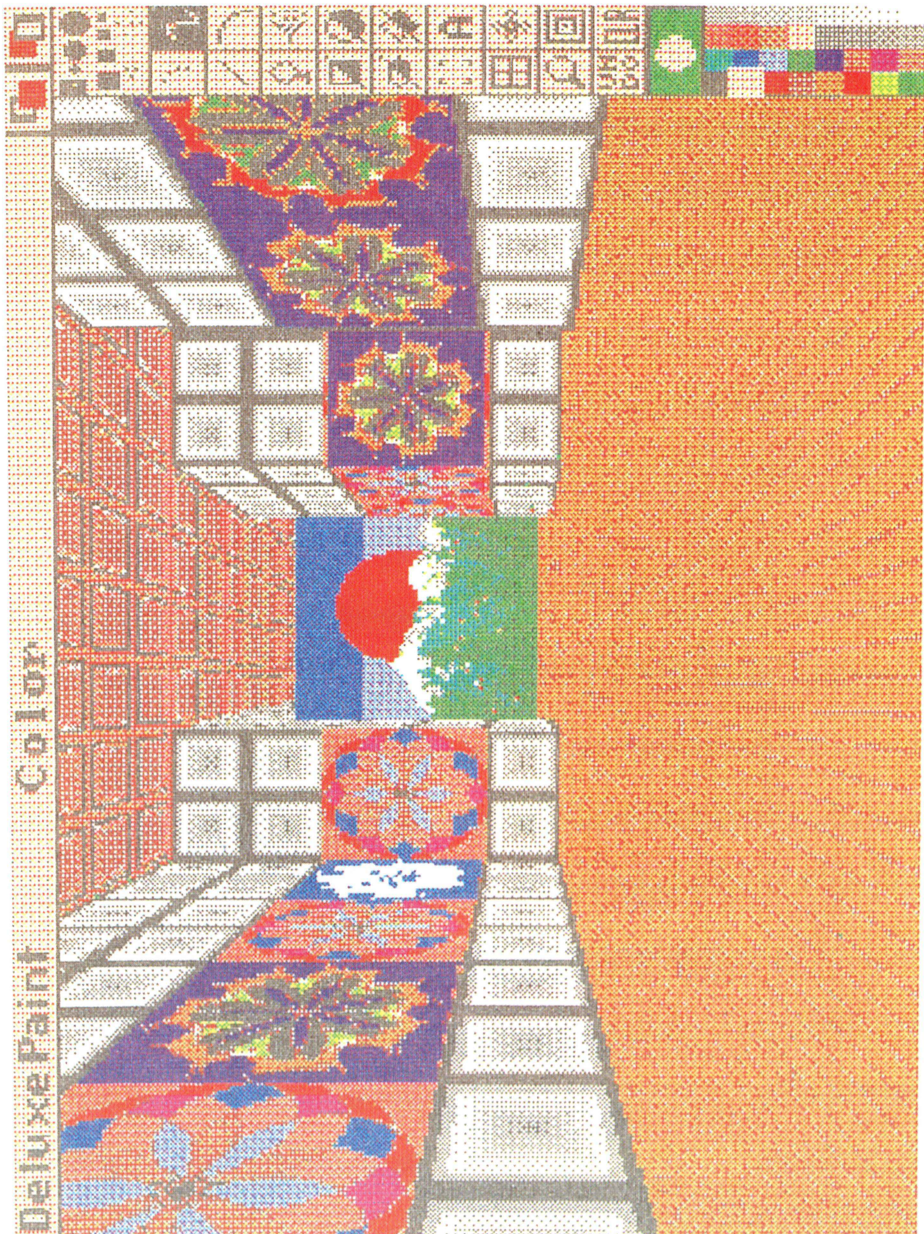



Abb. 4.3.1 Deluxe Paint II (Electronic Art Beispiel)

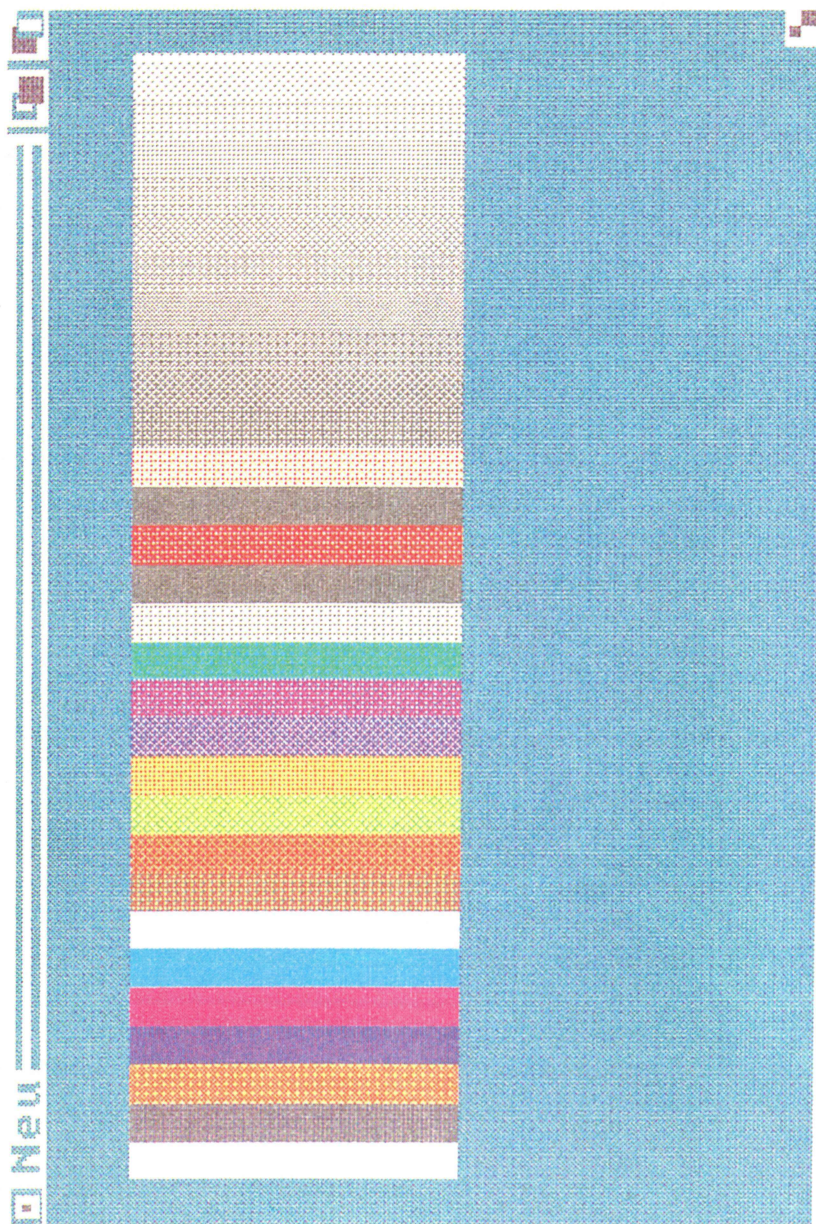



Abb. 5.2.1 32 Farben im Bildschirm

OBJECT.X objekt,x
OBJECT.Y objekt,y

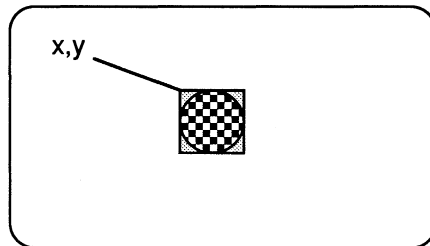



Abb. 6.2.2
OBJECT-Koordinaten

Abb. 6.2.3
OBJECT Ein-Aus

OBJECT.ON
OBJECT.ON objekt,...
OBJECT OFF
OBJECT OFF objekt,...

Der erste Parameter bei OBJECT.SHAPE ist eine Zahl größer Null und dient der Kennung des Objekts, so daß man später wieder darauf zurückgreifen kann. Wenn man mehr als ein Objekt braucht, und dieses genauso aussieht wie ein schon definiertes, so kann man dies z.B. Mit OBJECT.SHAPE 2,1 von 1 auf 2 kopieren. Dabei werden die Einzelpunkte nicht kopiert, sondern intern nur ein Verweis auf das erste Objekt gelegt.
Mit

**CLOSE 1**

sollte man dann die Datei wieder schließen. Wenn man das Programm so ausführt, passiert noch nichts weiter. Klar, wir haben ja auch noch nicht angegeben, wo das Objekt erscheinen soll. Dazu zeigt *Abb. 6.2.2* weitere Befehle.

**OBJECT.X 1,20**
**OBJECT.Y 1,100**

Auch jetzt würden die Objekte noch nicht sichtbar sein, man muß sie mit einem weiteren Befehl erst einschalten. *Abb. 6.2.3* zeigt die Definition des Ein- und Ausschaltbefehls. Dabei werden mit OBJECT.ON alle definierten Objekte eingeschaltet und mit OBJECT.ON objekt nur eines oder mehrere bestimmte, wenn man diese alle durch Kommas getrennt nacheinander aufführt. Mit OBJECT.OFF kann man sie wieder ausschalten.

**OBJECT.ON**

Nach diesem Befehl ist das Objekt sichtbar. Wenn man dieses Programm ausführt, so bleibt das Objekt auch nach der Rückkehr in den Basic-Interpreter aktiv, besser schreibt man also:

**WHILE MOUSE(0)**
**WEND**
**OBJECT.OFF**

und schließlich noch

**OBJECT.CLOSE**

gemäß der Syntax von *Abb. 6.2.4* , denn dadurch wird auch der belegte Speicherplatz wieder freigegeben. Nach dem OBJECT.CLOSE muß man im Programm das Objekt erst wieder mit OBJEKT.SHAPE definieren. Nach einem OBJECT.OFF genügt ein OBJECT.ON, um es wieder sichtbar zu machen. Nun wollen wir das Objekt mal über den Bildschirm bewegen. Dazu muß man nur in der Schleife die x- und y-Koordinate verändern. Z.B. nehmen wir mal die aktuellen Mauskoordinaten dafür und schreiben eine neue Schleife:

```
WHILE (MOUSE(0)=0) OR (MOUSE(1)  < 600)
 IF MOUSE(0)<0 THEN
  OBJECT.X 1,MOUSE(1)
  OBJECT.Y 1,MOUSE(2)
 END IF
WEND
```

Wenn Sie die linke Maustaste drücken, so folgt der Ball der Maus. Wenn Sie die linke Maustaste ganz rechts im Fenster drücken (x>600) , so wird das Programm beendet.

Bewegungen kann man aber vom Amiga-Basic auch automatisch durchführen lassen. *Abb. 6.2.5* zeigt die Syntax. Dazu gibt man die Geschwindigkeit des Objekts in x- und y-Richtung an. Dabei wird die Geschwindigkeit in Pixel pro Sekunde angegeben.

Bevor das Objekt aber automatisch bewegt wird, muß man einen weiteren Befehl angeben. *Abb. 6.2.6* zeigt die Syntax des OBJECT.START und

```
OBJECT.CLOSE
OBJECT.CLOSE objekt,....
```

Abb. 6.2.4 OBJECT schließen

```
OBJECT.VX objekt,geschw
OBJECT.VY objekt,geschw
```

Geschwindigkeit in Bildpunkten pro Sekunde

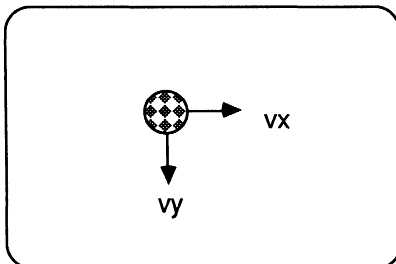



Abb. 6.2.5 OBJECT Geschwindigkeit

```
OBJECT.START
OBJECT.START objekt,...
OBJECT.STOP
OBJECT.STOP  objekt,...
```

Abb. 6.2.6 OBJECT Start und Stop

OBJECT.STOP-Befehls. Erst nach diesem Befehl erfolgt die Bewegung. Schreiben wir ein kleines Testprogramm:

```
OPEN "ball" FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,0
OBJECT.Y 1,100
OBJECT.VX 1,100
OBJECT.VY 1,0
OBJECT.ON
OBJECT.START
WHILE MOUSE(0)=0
WEND
OBJECT.STOP
OBJECT.OFF
OBJECT.CLOSE
```

Wenn man das Programm startet, so bewegt sich der Ball von links nach rechts über den Bildschirm. Rechts angekommen, bleibt er stecken. Dort ereignet sich nämlich eine Kollision mit dem rechten Rand. Diesen Umstand kann man auch abfragen und in eigenen Programmen verwenden. Dazu gibt es den Befehl ON COLLISION GOSUB unterprogramm, sowie die Befehle COLLISION ON usw., wie in *Abb. 6.2.7* zu sehen. Ferner finden Sie hier die Syntax des Befehls OBJECT.HIT, mit dem man auch einstellen kann, welche Objekte mit welchen kollidieren sollen (siehe Amiga-Basic-Handbuch). Hier wird wieder die Unterbrechnungsfähigkeit ausgenutzt und es lassen sich damit elegante Programme schreiben. Ergänzen wir unser Programm, so daß der Ball an der rechten Ecke abprallt und nach links zurückkommt und umgekehrt.Dazu braucht man auch noch die Funktion COLLISION(n). Mit COLLISION(0) liefert die Funktion das Objekt, mit dem die Kollision stattgefunden hat. COLLISION(-1) liefert das Fenster, in dem die Kollision stattfand und COLLISION(n) mit einem Wert größer 0 , also der Objektnummer liefert die Kennung des anderen Objekts mit dem das Objekt kollidiert ist. Ein Wert zwischen -1 und -4 gibt die Kollision mit dem Rand an. Dabei ist -1 die obere Begrenzung, -2 steht für die linke Begrenzung, -3 die untere Begrenzung und -4 die rechte Begrenzung. Der Aufruf der Funktion mit einem Wert größer Null entfernt außerdem den Kollisionshinweis aus einer internen Warteschlange, Sie muß also unbedingt aufgerufen werden.

```
OBJECT.HIT  objekt
OBJECT.HIT  objekt,selbstmaske
OBJECT.HIT  objekt,selbstmaske,stoßmaske
```

Abb. 6.2.7
Kollisions-Befehle

```
ON COLLISION GOSUB uprg
COLLISION ON
COLLISION STOP
COLLISION OFF
```

```
OPEN "ball" FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,20
OBJECT.Y 1,100
OBJECT.VX 1,100
OBJECT.VY 1,0
ON COLLISION GOSUB abprallen
OBJECT.ON
OBJECT.START
COLLISION ON
WHILE MOUSE(0)=0
WEND
OBJECT.STOP
OBJECT.OFF
OBJECT.CLOSE
END

abprallen:
 i = COLLISION(0)
 IF i = 0 THEN RETURN
 j = COLLISION(i)  : REM loeschen
 IF OBJECT.VX(1) > 0 THEN
  OBJECT.VX 1,-OBJECT.VX(1)
 ELSE
  OBJECT.VX 1,-OBJECT.VX(1)
 END IF
 OBJECT.START : REM neu starten
RETURN
```

Abb. 6.2.8
Ball-Programm

*Abb. 6.2.8* zeigt das komplette Programm.
Wenn man das Programm startet, so bewegt sich der Ball zunächst von links nach rechts, dort wird er aber jetzt reflektiert und bewegt sich wieder nach links, dann prallt er dort auch ab usw.
Wenn man die Grenzen für die Kollision festlegen will, so gibt es auch einen Befehl, der in *Abb. 6.2.9* abgebildet ist. Dazu gibt man einfach die linke obere und die rechte untere Ecke des Bereichs an.

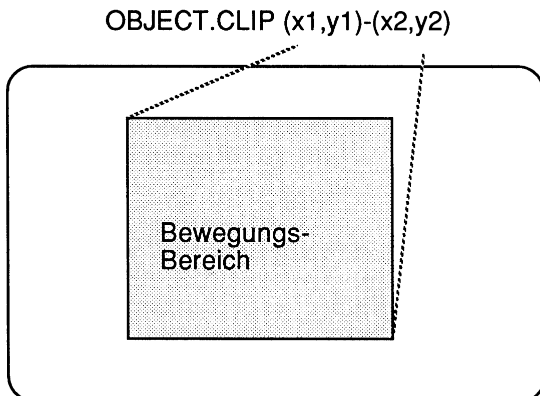



Abb. 6.2.9
OBJECT Feldbegrenzung

OBJECT.AX objekt,wert
OBJECT.AY objekt,wert

Beschleunigung in Bildpunkten pro Sekunde Quadrat

Abb. 6.2.10
OBJECT-Beschleunigung

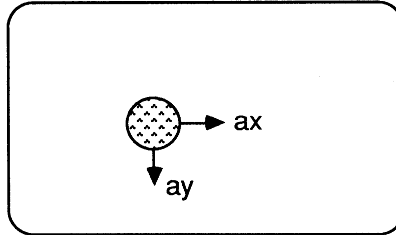
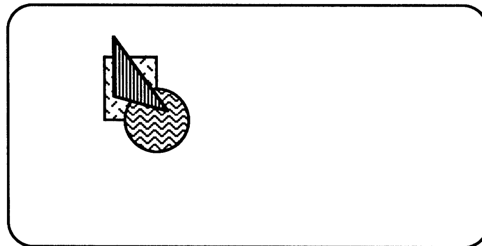



Abb. 6.2.11
OBJECT-Ebenen

OBJECT.PLANES objekt
OBJECT.PLANES objekt,bitebene,ebeneeinaus

OBJECT.PRIORITY objekt,prior

Abb. 6.2.12
OBJECT-Prioritäten



Einen weiteren Befehle zeigt *Abb. 6.2.10* , mit dem man auch beschleunigte Bewegungen durchführen kann.
Mit dem Befehl aus *Abb. 6.2.11* kann man die Zuordnung der Farben eines Objektes verändern und mit *Abb. 6.2.12* kann man die Anordnung der Objekte verändern. Bei diesen Befehlen möchte ich aber auf das Amiga-Basic-Handbuch verweisen.

## 6.3 Der hüpfende Ball

Wir wollen einmal ein kleines Simulationsprogramm schreiben. Ein Ball soll herunterfallen und dabei am Boden abprallen.
Dazu kann man das Programm aus Kapitel 6.2 modifizieren und ein erster Ansatz ist in *Abb. 6.3.1* sichtbar. Wenn Sie das Programm starten, so erscheint der Ball

```
OPEN "ball" FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,200
OBJECT.Y 1,10
OBJECT.VX 1,0
OBJECT.VY 1,0
OBJECT.AX 1,0
OBJECT.AY 1,2
ON COLLISION GOSUB abprallen
OBJECT.ON
OBJECT.START
COLLISION ON
WHILE MOUSE(0)=0
WEND
OBJECT.STOP
OBJECT.OFF
OBJECT.CLOSE
END

abprallen:
 i = COLLISION(0)
 IF i = 0 THEN RETURN
 j = COLLISION(i) : REM loeschen
 IF OBJECT.VX(1) > 0 THEN
  OBJECT.VY 1,-OBJECT.VY(1)
 ELSE
  OBJECT.VY 1,-OBJECT.VY(1)
 END IF
 OBJECT.START : REM neu starten
RETURN
```

Abb. 6.3.1
Hüpfender Ball

oben im Bild. Er fällt dann herunter und prallt unten im Bild ab. Der Ball führt dabei eine beschleunigte Bewegung durch. Das Abprallen wird dadurch realisiert, daß einfach das Vorzeichen der Geschwindigkeit umgedreht wird, so wie es beim elastischen Stoß auch der Fall ist.
Mit diesem Programm hüpft der Ball auf und ab, ohne daß die Bewegung aufhört, wie es in Wirklichkeit der Fall wäre. In Wirklichkeit verliert der Ball einmal durch Reibung mit der Luft ständig an Energie und zum Anderen geht beim Aufprall einiges an Energie verloren. Doch auch dies kann man leicht im Programm einbauen. Den Energieentzug durch Luftreibung können wir vernachlässigen. Beim Aufprall brauchen wir nur die Geschwindigkeit, mit der der Ball reflektiert wird, zu veringern, schon haben wir den gewünschten Effekt.
Schön wäre es auch noch, wenn der Ball nur in einem begrenzten Rahmen hüpfen kann, dazu kann man den OBJECT.CLIP-Befehl verwenden.
*Abb. 6.3.2* zeigt die neue Programmversion. Wenn Sie beide Programme vergleichen, so entdecken Sie neben den neuen Befehlen auch noch ein paar wesentliche Änderungen im Unterprogramm "abprallen". Vorher haben wir nicht geprüft mit welcher Seite der Ball kollidierte, also ob oben oder unten. Hier steht

```
OPEN "ball" FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,200
OBJECT.Y 1,40
OBJECT.VX 1,0
OBJECT.VY 1,0
OBJECT.AX 1,0
OBJECT.AY 1,2
OBJECT.CLIP (20,30)-(300,160)
LINE (20,30)-(300,168),1,b
ON COLLISION GOSUB abprallen
OBJECT.ON
OBJECT.START
COLLISION ON
WHILE MOUSE(0)=0
WEND
OBJECT.STOP
OBJECT.OFF
OBJECT.CLOSE
END

abprallen:
 i = COLLISION(0)
 IF i = 0 THEN RETURN
 j = COLLISION(i) : REM loeschen
 IF (j = -3) AND (OBJECT.VY(1)>0) THEN
  OBJECT.VY 1,-OBJECT.VY(1)+1
 ELSEIF (j = -1) AND (OBJECT.VY(1)<0) THEN
  OBJECT.VY 1,-OBJECT.VY(1)+1
 END IF
 OBJECT.START : REM neu starten
RETURN
```

Abb. 6.3.2
Hüpfender Ball mit
Begrenzung

die Abfrage IF (j=-3) AND (OBJECT.VY(1)<0) THEN ..., und damit wird dieser Teil nur ausgeführt, wenn der Ball eine Bewegung von oben nach unten ausführt und unten kollidiert. Entsprechend beim ELSE IF-Teil, der allerdings bei unserem Beispiel nicht wirksam wird, da der Ball nie an die obere Begrenzung kommt.

Aufgaben:

1. Erweitern Sie das Programm, so daß der Ball auch eine horizontale Geschwindigkeitskomponente bekommt (Hinweis: keine beschleunigte Bewegung). Die Bewegung muß aber auch gebremst werden, sobald der Ball Bodenkontakt bekommt.
2. Erweitern Sie das Programm, so daß der Ball an der rechten bzw. linken Begenzung abprallen kann.
3. Wie kann man den Ball stärker bremsen, was muß man tun, damit der Ball am Schluß auch völlig zur Ruhe kommt?
4. Geben Sie eine Meldung aus, wenn der Ball am Boden liegt.

## 6.4 Mondlandung mit Fähre

Eine Mondlandung soll mit Hilfe einer Fähre durchgeführt werden. Das Fährenmodell soll ein Sprite sein, das Sie z.B. mit dem OBJEDIT erstellen können. Wählen Sie dabei den Bobs-Mode, denn Sprites haben auch bei der Positionierung besondere Eigenschaften und außerdem sind Sie dann nicht so eingeschränkt.
Wenn man die linke Maustaste drückt, so soll die Fähre gebremst werden können. eine saubere Landung erfolgt nur dann, wenn die Fähre mit einer minimalen Geschwindigkeit aufsetzt. Dabei kann man einige Teile des Programms aus 6.3 verwenden.
Im Hauptprogramm muß nun eine Abfrage eingebaut werden, die falls die Maustaste gedrückt wird, eine Beschleunigung weg von der Oberfläche durchführt. Die Landung ist beendet, wenn man entweder abgestürzt ist oder eine saubere Landung durchgeführt hat. Die Entscheidung, ob die Landung geglückt ist, kann man bei der Kollision durchführen. Dazu wird die Geschwindigkeit in y-Richtung abgeprüft.
*Abb. 6.4.1* zeigt das fertige Programm. Wenn man die linke Maustaste drückt, so wird der Wert 1 von der Geschwindigkeit in y-Richtung abgezogen. Dabei wird zuvor noch eine Warteschleife ausgeführt, so daß man bei dauerndem Drücken der Maustaste keine zu starke Bremswirkung erzielen kann.
Neben der Bruchlandung gibt es hier noch eine andere Fehlermeldung. Wenn man

```
OPEN "faehre" FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,200
OBJECT.Y 1,40
OBJECT.VX 1,0
OBJECT.VY 1,0
OBJECT.AX 1,0
OBJECT.AY 1,1
OBJECT.CLIP (20,30)-(300,160)
LINE (20,30)-(300,160),1,b
ON COLLISION GOSUB gelandet
OBJECT.ON
OBJECT.START
COLLISION ON
ende = 0
WHILE ende = 0
 FOR i=1 TO 70: NEXT i: REM warten
 IF MOUSE(0)<0 THEN
  OBJECT.VY 1,OBJECT.VY(1)-1
 END IF
WEND
OBJECT.STOP
FOR i=1 TO 1000: NEXT i: REM warten
OBJECT.OFF
OBJECT.CLOSE
END
```

```
gelandet:
 i = COLLISION(0)
 IF i = 0 THEN RETURN
 j = COLLISION(i) : REM loeschen
 IF (j = -3) AND (OBJECT.VY(1)>=0) THEN
  IF OBJECT.VY(1) < 10 THEN
   PRINT "gelandet."
  END IF
  IF OBJECT.VY(1) >= 10 THEN
   PRINT "Bruchlandung!!!"
  END IF
  PRINT "Landegeschwindigkeit = ";OBJECT.VY(1)
 ELSEIF (j = -1) AND (OBJECT.VY(1)<0) THEN
  PRINT "Faehre verlaesst das Orbit"
  OBJECT.OFF
 END IF
 ende = 1
RETURN
```

Abb. 6.4.1
Mondlandung

mit der oberen Grenze kollidiert, dann soll die Fähre den Orbit verlassen.
Aufgaben:
1. Begrenzen Sie den Treibstoff. Dies kann man dadurch machen, daß man einen Zähler einführt, der immer dann hochgezählt wird, wenn man die Maustaste drückt und damit bremst. Wenn der Zähler einen bestimmten Wert überschritten hat, so soll eine Fehlermeldung erfolgen.
2. Erweitern Sie das Programm, so daß man auch die x-Richtung der Fähre beeinflussen kann. Dazu muß aber die Steuerung geändert werden. Man kann dazu z.B. Schalterfelder verwenden, die man rechts im Bild anordnet und damit jeweils den Schub nach links, rechts oder oben steuern kann.

# 7 Töne und Geräusche

Töne und Geräusche sind eine besondere Stärke des Amiga. Kaum ein Spielprogramm ist heute ohne Ton denkbar. Daher werden wir uns auch kurz mit diesen Fähigkeiten beschäftigen. Dabei gibt es auch hierfür wieder einige besondere Befehle, mit denen man leicht Töne und Geräusche erzeugen kann.

## 7.1 Die Tonleiter

Zur Programmierung von Tönen und Geräuschen muß man nur wenige Befehle kennen. Der Amiga erzeugt Töne und Geräusche ähnlich wie bei einem Tonband. Sie befinden sich dazu in digitalisierter Form im Speicher. Die Wellenform wird im Speicher in Form von Zahlen festgehalten.
Der Befehl SOUND dient primär der Tonerzeugung. *Abb. 7.1.1* zeigt die Syntax. Die einfachste Form des Befehls lautet SOUND frequenz, dauer. Dabei wird die Frequenz in Herz angeben. Der Wert von "dauer" darf zwischen 0 und 77 liegen. Dabei entspricht der Wert 18.2 ca. einer Sekunde.
Probieren Sie doch gleich mal den Befehl

**SOUND 1000,20**

aus. Ein 1-kHz-Ton ist dann zu hören. Sound kann aber noch mehr. Man kann eine Lautstärke angeben, der Wert von "lautstärke" liegt im Bereich 0 (kleinste Lautstärke) und 255 (maximale Lautstärke). Wenn man den Wert nicht angibt, ist ein Wert von 127 voreingestellt.

```
FOR laut=0 TO 255
  SOUND 500,1,laut
NEXT laut
```

Nach Start schwillt ein 500 Hz-Ton langsam auf maximale Lautstärke an. Schließlich kann man bei dem Befehl auch noch einen Kanal angeben. Der Amiga besitzt insgesamt 4 Kanäle, die von 0 bis 3 gezählt werden. Dabei sind zwei Stereo-Kanäle vorhanden. Kanal 0 oder 3 ist dem linken Stereo-Kanal zugeordnet, Kanal 1 und 2 dem rechten.

```
SOUND frequenz,dauer
SOUND frequenz,dauer,lautstärke
SOUND frequenz,dauer,lautstärke,kanal

SOUND WAIT
SOUND RESUME
```

Abb. 7.1.1
Der SOUND-Befehl

Dazu ein kleiner Test, der alle Kanäle anspricht:

```
SOUND 500,60,127,0
SOUND 510,60,127,1
SOUND 520,60,127,2
SOUND 530,60,127,3
```

Die einzelnen Frequenzen sind dabei mit Absicht nicht ganz identisch, so daß sich ein interessanter Schwebungseffekt einstellt.
Wenn man mehrere Kanäle wirklich und synchron gleichzeitig starten will, so braucht man einen neuen Befehl. SOUND WAIT hält alle nachfolgenden SOUND-Befehle solange in einer Warteschlange, bis man sie mit SOUND RESUME freigibt.
Beispiel:

```
FOR i=1 TO 10
  SOUND WAIT
  SOUND 510,1,127,0
  FOR j=1 TO 500 : NEXT j
  SOUND 520,1,127,1
  FOR j=1 TO 500 : NEXT j
  SOUND RESUME
NEXT i
```

Durch das Programm werden 10 einzelne Töne erzeugt. Wenn Sie einmal in diesem Programm die Befehle SOUND WAIT und SOUND RESUME weglassen, so hören Sie 20 mal den Ton.
Wie kommt man nun zu einer Tonleiter? Es gilt dabei folgendes: von Oktave zu Oktave verdoppelt sich die Frequenz. Die 12 Töne einer Oktave erhält man, indem man den Startwert nacheinander mit der 12ten Wurzel aus 2 multipliziert. Ein kleines Programm gibt eine Oktave aus:

```
freq = 440
FOR i=1 TO 12
  PRINT freq
  SOUND freq,10
  freq = freq * 1.059463094
NEXT i
```

Nach C-Dur hört sich das natürlich nicht an, denn dort darf man nicht alle Halbtöne mitspielen. Das folgende Programm spielt dann eine Tonleiter.

```
FOR i=1 TO 8
  READ freq
  SOUND freq,10
NEXT i
DATA 523.25,587.32,659.25
DATA 698.45,783.98,880,987.766
DATA 1046.5
```

Nun steht den großen Musikprogrammen ja nichts mehr im Wege.

## 7.2 Ein einfaches Musikprogramm

Wir wollen eine Klaviatur bauen, die man mit der Maus bedienen kann.
Die Klaviatur soll über ein paar Oktaven verfügen. *Abb. 7.2.1* zeigt das fertige Programm.

```
REM kleines Musikprogramm
 REM weisse Tasten
 FOR i=0 TO 31
  x = i * 20
  LINE (x,100)-(x+18,150),1,bf
 NEXT i
 REM schwarze Tasten
 FOR i=0 TO 29
  x = i * 20 + 12
  IF ((i MOD 7) <> 2) AND ((i MOD 7) <> 6) THEN
   LINE (x,100)-(x+12,130),2,bf
  END IF
 NEXT i
 REM berechnen der Frequenzen
 DIM f(90)
 f0 = 130.81
 FOR i = 0 TO 90
  f(i) = f0
  f0 = f0 * 1.059463094#
 NEXT i
 DIM sw(7),ws(7)
 FOR i=0 TO 6
  READ sw(i)
 NEXT i
 FOR i=0 TO 6
  READ ws(i)
 NEXT i
 REM Mausabfrage
 WHILE 1=1
  FOR i=1 TO 25 : NEXT i
  IF MOUSE(0) < 0 THEN
   x = MOUSE(1)
   y = MOUSE(2)
   IF (y>100) AND (y<130) THEN
    i1 = (x-12) \ 20
    offset = ((x-12) \ 140)*12
    IF sw(i1 MOD 7) <> -1 THEN
     SOUND f(sw(i1 MOD 7)+offset),1
    END IF
   ELSEIF (y>=130) AND (y<150) THEN
    i2 = x \ 20
    offset = (x \ 140) * 12
    SOUND f(ws(i2 MOD 7)+offset),1
   END IF
  END IF
 WEND
 DATA 1,3,-1,6,8,10,-1
 DATA 0,2,4,5,7,9,11
```

Abb. 7.2.1
Das kleine Musik-Programm

Zunächst werden im Programm die weißen Tasten gezeichnet. Dabei haben die Tasten einen Abstand von 20 Punkten. Anschließend werden die schwarzen Klaviaturtasten darüber gezeichnet. Dabei ist zu beachten, daß bei einer Klaviatur bestimmte Tasten fehlen. Es sind dies die 3 und 7 jeder Oktave. Im Programm wird dies durch eine IF-Anweisung abgefragt.
Da immer sieben weiße Tasten eine Oktave umfassen, kann man mit Modulo 7 eine einfache Regel aufstellen.
Als nächstes folgt die Berechnung der Frequenzen. Dabei soll die tiefste Frequenz 130.81 sein, die auf der Taste C liegt.
Wenn man später bestimmen will, welche Taste zu welchem Ton gehört, so kann man sich eines Tricks bedienen. Dazu werden nun zwei Felder eingelesen. Das Feld "sw" enthält den Index der ersten 7 schwarzenTasten, einschließlich der nicht vorhandenen. Die Tasten, die auf der Klaviatur nicht vorkommen, sind durch den Wert -1 gekennzeichnet.
Im Feld "ws" sind entsprechend die weißen Tasten vermerkt. Dabei genügt es, die Felder für sieben Tasten vorzusehen, da sich das Schema danach wiederholt.
Nun folgt die Mausabfrage. Eine kleine Warteschleife sorgt dafür, daß die Mausabfrage nicht zu häufig vorgenommen wird, sonst hält der Ton später zu lange an, denn der SOUND-Befehl merkt sich den Aufruf im voraus.
Wenn der Mauszeiger im oberen Bereich der Klaviatur war, so sind die schwarzen Tasten betroffen, im unteren Bereich sind es die weißen Tasten.
Wurde eine Taste gedrückt, so wird nun der Index in das Hilfsfeld "sw", bzw. "ws" berechnet. Die Variable "offset" enthält die Oktave und nimmt Werte von 0,12,24 usw. an. *Abb. 7.2.2* zeigt die Klaviatur, wie sie vom Programm ausgegeben wird.
Aufgaben:
1. Erweitern Sie das Programm, so daß beim Anklicken einer Taste diese invers (z.B. Rot) aufleuchtet.
2. Erweitern Sie das Programm, so daß eine Melodiefolge gespeichert wird, damit man sie später, z.B. durch einen Menüpunkt, wieder abspielen kann.

## 7.3 Geräusche

Einen Befehl hatten wir im Zusammenhang mit dem SOUND-Befehl noch nicht besprochen, den WAVE-Befehl. Damit kann man eigene Wellenformen definieren und mit dem SOUND-Befehl verwenden.
*Abb. 7.3.1* zeigt die Syntax. Um den Befehl zu verwenden, muß man ein Ganzzahlen-Feld mit mindestens 256 Elementen definieren. Darin speichert man die Punkte der Wellenform. Diese müssen dabei im Bereich -128..127 liegen.
Beispiel:

```
DIM welle%(256)
FOR i=0 TO 255
 welle%(i) = i-256
NEXT i
WAVE 0,welle%
SOUND 1000,30
```

Es handelt sich dabei um eine sägezahnförmige Schwingung, die härter klingt als

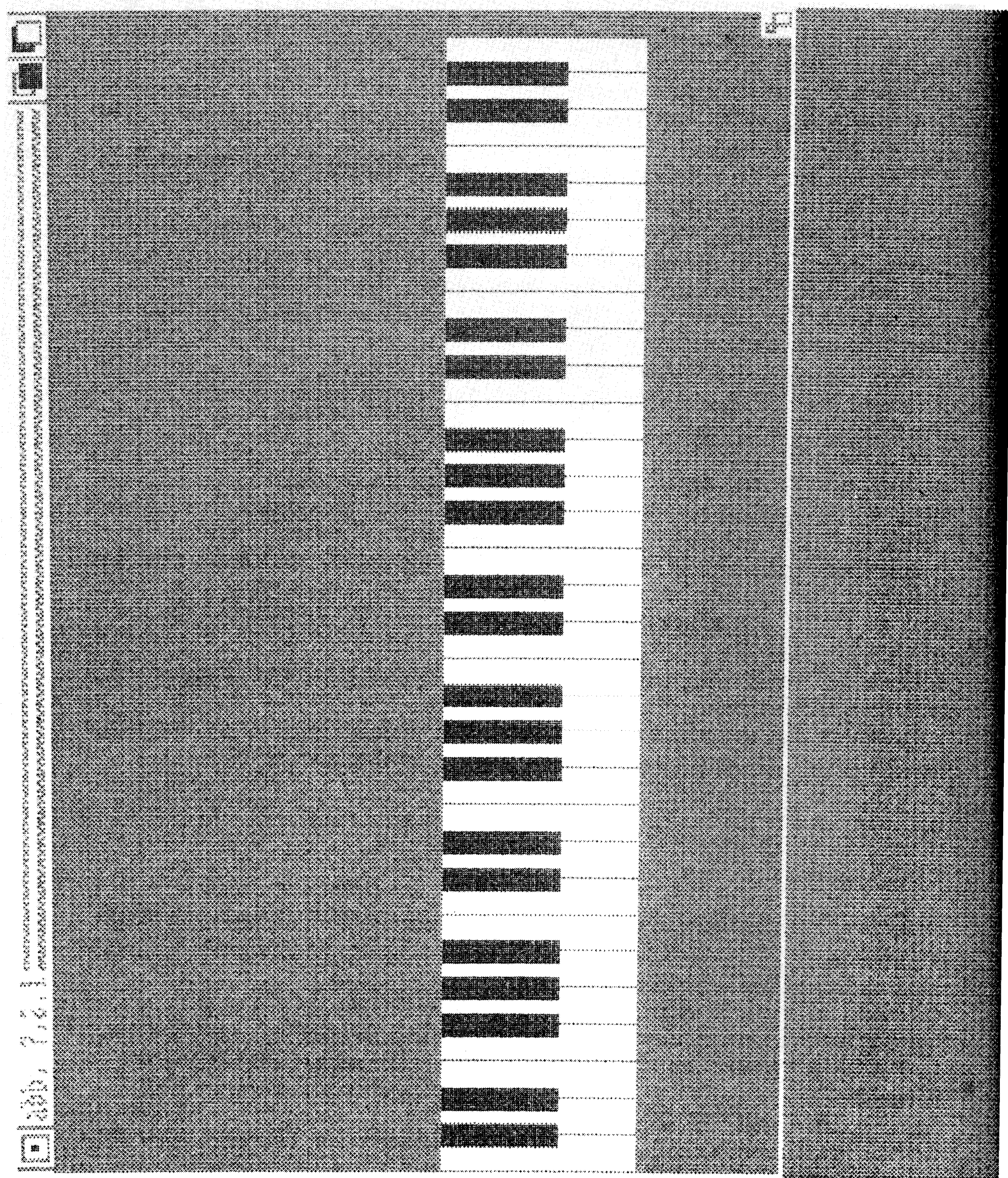



Abb. 7.2.2 Bildschirmausdruck des Musikprogramms

WAVE kanal,definitionsfeld

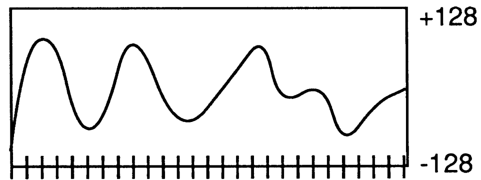



Abb. 7.3.1
Der WAVE-Befehl

Das Feld mit dem Namen SIN ist voreingestellt. Sonst muß man ein Ganzzahlenfeld als Parameter angeben.

eine reine Sinus-Schwingung, wie sie voreingestellt ist. Mit WAVE 0,SIN kann man übrigens die Sinusschwingung einstellen.
Um auch zischende Geräusche erzeugen zu können, braucht man häufig ein Rauschen. Dies kann man z.B. mit folgendem Programm erzeugen:

```
DIM welle%(256)
FOR i=0 TO 255
 welle%(i) = rnd(1)*256-128
NEXT i
WAVE 0,welle%
SOUND 20,30
```

So ganz nach reinem Rauschen klingt das nicht, dazu müßte man ein größeres Feld verwenden. Übrigens kann man das Feld mit **ERASE welle%** nach Aufruf des WAVE-Befehls wieder löschen und damit Speicherplatz sparen.

# 8 Sprache

Der Amiga hat im Betriebssystem ein praktisches Programm eingebaut, mit dem man ihn zum Sprechen bringen kann.
Das Verfahren beruht auf der sogenannten Phonemsysthese. Dabei sind einzelne Phoneme im Computer gespeichert, sie wurden zuvor mit einem Digitalisierer von einem menschlichen Sprecher erzeugt.
Der Befehl SAY erlaubt es, Phoneme auszugeben. Damit man nicht selbst Texte in Phoneme zerlegen muß, gibt es die Funktion TRANSLATE$, die einen Textstring in Phoneme umwandelt.
Probieren Sie einmal:

**SAY TRANSLATE$("Hallo wie geht es")**

Das klingt natürlich sehr amerikanisch, kein Wunder, denn sowohl der Phonemsatz als auch die Translate-Funktion sind für die englische Sprache gedacht. SAY kann noch einen zusätzlichen Parameter haben, ein Ganzzahl-Feld, in dem verschiedene einzelne Parameter stehen. Das Feld muß mindestens 9 Elemente besitzen. Damit kann man verschiedene Betonungsarten usw. einstellen. Uns soll jedoch hier die Grundform genügen, da man damit die besten Resultate erhält. Wer übrigens einen Hinweis sucht, wie man deutsche Texte besser sprechen lassen kann, der sei auf den Anhang des Amiga-Basic-Handbuchs verwiesen.

## 8.1 Ein Programm zum Vorlesen von Listings

Wir wollen im Computer gespeicherte Listings vorlesen lassen. Das ist zuweilen ganz praktisch, wenn man ein abgetipptes Listing vergleichen will. Man läßt sich dann das Eingetippte einfach vom Computer vorlesen und vergleicht es mit der Vorlage, auf die man sich dann ganz konzentrieren kann. Das Listing muß man allerdings in ASCII abgespeichert haben, z.B. durch SAVE "name",A.
*Abb. 8.1.1* zeigt das Programm.
Sobald allerdings deutsche Wörter im Text vorkommen wird man seine Schwierigkeiten mit dem Verstehen haben, die Basic-Befehlswörter hingegen versteht man recht gut.

```
REM Vorlesen von Listings
INPUT "Name der Datei";d$
OPEN d$ FOR INPUT AS 1
WHILE NOT EOF(1)
 LINE INPUT #1,zeile$
 PRINT zeile$
 SAY TRANSLATE$(zeile$)
WEND
CLOSE 1
END
```

Abb. 8.1.1
Das Programm "Listings vorlesen"

# 9 Literatur


1. Amiga-BASIC. Commodore Handbuch im Lieferumfang des Amiga enthalten.
2. Michael Boom, THE AMIGA, Microsoft Press 1986.
3. Rügheimer - Spanik, AMIGA BASIC, Data Becker GmbH 1986.
4. Weltner - Hornig, AMIGA TIPS & TRICKS, Data Becker GmbH 1986.
5. Amiga Hardware Reference Manual, Addison Wesley 1986.
6. Amiga ROM Kernel Reference Manual: Library and Devices, Addison Wesley 1986.
7. Amiga ROM Kernel Reference Manual: Exec, Addison Wesley 1986.
8. Amiga Intuition Reference Manual: Library and Devices, Addison Wesley 1986.

# 10 Terminologieverzeichnis

## A

**Access**
Zugriff; Zugriff zum Beispiel auf einen Speicher.

**Ada**
Eine Programmiersprache, die im Auftrag des amerikanischen Verteidigungsministerium entwickelt wurde. Ada ist eine sehr umfangreiche Programmiersprache, die auch Multitasking-Konzepte usw. enthält.
Beispiel:

```
loop
  select
    accept READ( X: out ITEM) do
      X := STORED ;
    end READ ;
  or
    accept WRITE(X : ITEM) do
      STORED := X ;
    end WRITE ;
  end select ;
end loop ;
```

**Adresse**
Eine Bezeichnung für einen bestimmten Speicherplatz oder Speicherbereich.

**Adreßbus**
Ein Bus auf den die Adreßleitungen eines Mikroprozessors geführt werden.

**Akkumulator**
Ein Register, in dem arithmetische und logische Operationen ausgeführt werden können.

**Algol**
Algorithmic Language. Es handelt sich um eine Programmiersprache für den technisch wissenschaftlichen Bereich.
Programmbeispiel:

```
'BEGIN'
  'REAL' alpha, betha;
  alpha := 3.1;
  INREAL(1,betha);
  OUTREAL(2,betha*alpha);
'END'
```

**ALU**
Arithmetic Logic Unit, Rechenwerk. In diesem Teil des Rechners werden die arithmetischen und logischen Verknüpfungen ausgeführt.

**APL**
A Programming Language. Eine Programmiersprache für den technisch wissenschaftlichen Bereich, die Sprache verwendet dabei spezielle Zeichen. Beispiel:

```
    ∇ R←X    WL    Y
[1]   ⍝ Waagrechte Linie einfuegen
[2]   R← (~R ε X)/R←⍋ (⌊1↑ρ Y),X
[3]   R← (Y,[⎕ IO]´-´)[(⎕ IO+1↑ρ Y) ⌊ R;]
    ∇
```

**APU**
Arithmetic Processor Unit, siehe auch FPU

**ASCII**
American Standard Code for Information Interchange. Wird auch mit ISO-7-Bit-Code bezeichnet (DIN 66003). Codierungsart für Zeichen.

**Assembler**
Ein Übersetzungsprogramm, das aus einem mnemotechnischen Programm-Code einen Maschinen-Code erstellt

## B

**Bankselekt**
Unter einer Speicher-Bank versteht man zum Beispiel eine Gruppe von Speichern. Selekt steht für Auswahl. Bankselekt bedeutet also die Auswahl einer Gruppe von Speichern. Man verwendet ein Bankselekt-Signal zum Beispiel zum Erweitern des Adreßraums

**BAS-Mischer**
Aus dem Synchron- und Videosignal wird durch elektrisches Mischen ein Signal gewonnen, das beide Signale auf einer Leitung transportieren kann. Dieses BAS-Signal ist zur Ansteuerung von vielen Monitoren geeignet.

**Basic**
Beginners All Purpose Symbolic Instruction Code. Eine einfache Programmiersprache, die besonders auf Heimcomputern sehr verbreitet ist, jedoch den Nachteil besitzt, nicht strukturiert zu sein.
Beispiel:

```
10 PRINT "Quadratwurzeltabelle"
20 FOR I=1 TO 10
30 PRINT I,SQRT(I)
40 NEXT I
```

**Baudrate**
Messung des Datenflusses, wobei die Zeit zur Übertragung des kürzesten Elements als Maß genommen wird. Beispiel: 1200 Baud bedeuten eine Übertragung von 1200 Bit pro Sekunde

**Baudrate-Generator**
Ein Baustein zur Erzeugung eines Taktes, der dann für eine serielle Übertragung verwendet wird.

**Betriebssystem**
Eine Reihe von Programmen, die es dem Computer ermöglichen, selbstständig Programme zu bearbeiten, CP/M und MSDOS sind z.B. solche Betriebssysteme für Mikrorechner.

**Bi-Direktionale Bustreiber**
Ein Schaltkreis, der logische Informationen auf einer Leitung in beiden Richtungen übertragen kann.

**Bildwiederholspeicher**
Die Information, zur Darstellung auf dem Bildschirm wird im Bildwiederholspeicher bereit gehalten, so daß sie fortlaufend ausgegeben werden kann.

**Bit**
Binary Digit. Kleinste Informationseinheit

**Boot**
Das Neustarten eines Systems, bei dem Programme geladen werden, wird auch als Boot bezeichnet.

**Branch**
Verzweigung, Sprung

**Buffer**
Puffer; Speicher in dem Daten kurzzeitig festgehalten werden, oder auch Treiber zum Schalten von größeren Lasten.

**Bus**
Sammelleitung, an die mehrere Bausteine angeschlossen werden können. Dabei können auch mehrere Bausteine Daten auf den Bus angegeben, jedoch nicht zur gleichen Zeit. Eine Auswahllogik sorgt dafür, daß immer nur ein Busteilnehmer sendet, wobei jedoch alle weiteren hören dürfen.

**Byte**
8 Bits werden als 1 Byte zusammengefaßt

**C**
Eine Programmiersprache, die zum Beispiel mit dem CP/M68k-Betriebssystem geliefert wird. Die Sprache ähnelt sehr der Sprache Pascal. Programmbeispiel:

```
main()
{
  int i;
  float sqrt();
  for (i = 1; i<=10; i++) {
    printf(
      " Wurzel aus %d ist  %f",
      i,sqrt(i));
  }
}
```

**Cache**
Ein schneller Speicher, der z.B.in dem Prozessor-IC integriert ist, und es z.B. erlaubt kleine Programmschleifen schnell ausführen zu können. Der Prozessor 68020 besitzt z.B. einen Programm-Cache.

**Clock**
Takt

**Cobol**
Common Business Oriented Language. Eine Programmiersprache vorwiegend für kaufmännische Probleme. Beispiel:

```
PROCEDURE DIVISION.
START.
  MOVE ZEROS TO N
  MOVE 1 TO FAKULTAET.
  ACCEPT M-EIN FROM
          KARTEN-LESER;
  MOVE M-EIN TO M
  ...
```

**Comal**
Common Algorithmic Language; diese Sprache entstand 1973 aus einer Mischung von Basic und Pascal. Sie enthält daher strukturierte Elemente und Parametermechanismen
Programmbeispiel:

```
0010 PROC FENSTER(X,Y) CLOSED
0020   DIM LEERZ$ OF 40
0030   LEERZ$(1:40) := " "
0040   POSI(X,1)
0050   FOR ZN:=1 TO Y-X+1 DO PRINT
          LEERZ$
0060  POSI(X,1)
0070 ENDPROC FENSTER
```

**Compiler**
Ein Übersetzungsprogramm, das eine höhere Programmiersprache in den Maschinen-Code übersetzt. Siehe Kapitel Pascal/S und Gosi

**Conditional**
Bedingt

**Controller**
Steuereinheit

**CP/M**
Disk Operating System für 8080, 8085, Z80, 8086 und 68000 von DIGITAL RESEARCH. Siehe Kapitel Betriebssysteme

**Cross-Assembler**
Ein Assembler, der nicht auf der Maschine läuft, für die er Code erzeugt. Zum Beispiel ist ein Assembler für den 68000 ein Cross-Assembler, wenn man ihn unter CP/M80, also z.B. auf dem Z80 laufen lassen kann.

**Cross-Compiler**
Ein Übersetzer für eine höhere Programmiersprache, der auf einem anderen Prozessor läuft, als für welchen er Maschinen-Code erzeugt.

**CRT**
Cathode Ray Tube; Datensichtgerät oder Bildschirm

**Cursor**
Sichtmarke zur Kennzeichnung der aktuellen Schreibposition auf dem Bildschirm.

## D

**Datenbus**
Ein Bus, auf den die Datenleitungen eines Mikroprozessors geführt werden.
**Debugging**
Wörtlich "entwanzen". Gemeint ist die Fehlersuche und Beseitigung in Programmen.
**Decrement**
Erniedrigen, herunterzählen
**Digit**
Ziffer, Stelle
**DIL**
Dual In Line - Gehäuseform
**Direktory**
Inhaltsverzeichnis: z.B. von einer Diskette
**DMA**
Direct Memory Access; direkter Zugriff auf den Speicher eines Rechners, wobei die Zugriffssteuerung von einer Peripherieeinheit vorgenommen wird.
**DOS**
Disc Operating System, Betriebssystem; siehe Kapitel Betriebssystem
**dynamische Speicher**
Ein Speicher, bei dem die Speicherzellen ständig angesprochen werden müssen, damit sie ihre Information nicht verlieren.

## E

**Editor**
Ein Programm, das die Eingabe von Text erlaubt.
**EEPROM**
Electrical Erasable Programmable Read Only Memory. Ein Speicher, der sich elektrisch programmieren und löschen läßt. Dabei bleibt die Information nach dem Ausschalten der Versorgungsspannung erhalten. Im Gegensatz zu den EPROMs werden die EEPROMs durch Anlegen einer höheren Spannung gelöscht.
**Emulation**
Softwaremäßige Nachbildung eines Computers, so daß der Befehlssatz des einen Computers auf einem anderen verfügbar wird. Für den 68000/8 gibt es einen Z80-Emulator, so daß man Z80 Programme ablaufen lassen kann, obwohl man einen 68000/8 verwendet.
**Enable**
Freigabe
**enter**
Eingeben
**EPROM**
Erasable Programmable Read Only Memory. Ein löschbarer Nur-Lese-Speicher. Siehe Kapitel PROMMER
**erase**
Löschen
**Error**
Fehler, Irrtum
**Even**
bedeutet gerade im Gegensatz zu ungerade
**Expression**
Ausdruck

**Fan-in**
Eingangslastfaktor
**Fan-out**
Ausgangslastfaktor; er gibt an, wieviele Bausteine der gleichen Logikserie an diesem Ausgang angeschlossen werden dürfen.
**Festwertspeicher**
Ein Speicher, dessen Inhalt nicht (oder nur mit Mühe) geändert werden kann.
**Fifo**
First In First Out. Zuerst eingehende Daten werden auch zuerst wieder ausgegeben.
**File**
Datei, Daten. Eine Ansammlung von Datengruppen, die in einer Datei angelegt werden.
**Firmware**
Eine Software, die fest zur Funktionsfähigkeit eines Systems nötig ist und z.B. in einem ROM abgelegt ist.
**Fixed-Point**
Festkomma
**Flag**
Eine Marke oder ein Flip-Flop zum Festhalten eines Zustands.
**Floating-Point**
Gleitkomma
**Forth**
Eine Programmiersprache, die auf der UPN (umgekehrt polnische Notation) basiert.
Beispiel:

```
: TEXTAUS ."Hallo Forth Erg="
   + * . ;
3 4 5 TEXTAUS
```

**Fortran**
Formula Translation. Eine problemorientierte Programmiersprache für den technisch wissenschaftlichen Bereich. Beispiel:

```
      SUBPROGRAM BEISPIEL
       COMPLEX Z1,Z2
C     komplexe Zahlen sind möglich
      READ* A,B,C
      D = (B*B-4.*A*C)/(4.*A*A)
      IF (D.GE.0.) GOTO 20
      Z1 = COMPLX(-B/2.*A),SQRT(-D)
      Z2 = CONJG(Z1)
      PRINT*, 'Z1=',Z1,' Z2=',Z2
20    STOP
      END
```

**FPU**
Floating Point Unit. Gleitkommarechner. Für den 68020 und 68000/8 gibt es z.B. den Baustein 68881, der eine FPU darstellt.
**FSK**
Frequency shift. Verfahren bei der Aufzeichnung auf Datenträger.

## G

**Gate**
Verknüpfungsschaltung
**GND**
Masseanschluß, 0V

## H

**Handshake**
Quittungsbetrieb. Durch Steuersignale werden Geräte mit verschiedenen Arbeitsgeschwindigkeiten synchronisiert.
**Hardcopy**
Kopie. Zum Beispiel Ausdruck eines Bildschirminhalts
**Hardware**
Damit sind alle Bauteile, Geräte eines Systems gemeint,
**Hexadezimal**
Siehe Sedezimal
**High order**
Höherwertige Stelle

## I

**ICE**
In-Circuit Emulator. Gerät zum Test und Entwicklung von Mikrorechnerschaltungen
**Increment**
Erhöhen, raufzählen
**Initialisierung**
Die Anfangsschritte in einem Programm, um definierte Startwerte zu erhalten
**Input**
Eingabe
**Instruktionszyklus**
Ablauf eines Befehlsausführungsvorgangs
**INT**
Interrupt. Unterbrechungsanforderung
**Interpreter**
z.B. ein Programm, das Befehle einer höheren Programmiersprache direkt ausführt und sie nicht vorher in Maschinensprache übersetzt.
**Interrupt**
Unterbrechung

## J

**Job**
Auftrag
**Joystick**
Entweder ein Steuerknüppel mit vier Kontakten in den Endstellungen, oder ein Steuerknüppel mit zwei Potentiometern
**Jump**
Sprung

## K

**Keyboard**
Tastatur
**Kit**
Bausatz
**kompatibel**
Austauschbar, aneinander angepaßt;

## L

**Label**
Marke. In Programmiersprachen ist damit meist eine symbolische Adresse gemeint
**Lichtgriffel**
Ein Stift mit einem optischen Aufnehmer, der auf den Bildschirm gehalten wird. Der Rechner kann dann die Position des Lichtgriffels ermitteln.
**Lifo**
Last In First Out. Zuletzt gespeicherte Daten werden zuerst ausgegeben (Stack ).
**Linker**
Ein Programm, das mehrere Teilprogramme, die schon übersetzt wurden zu einem gesamten Programm zusammenfügen kann. Dabei können die Teilprogramme Bezüge untereinander enthalten
**Lisp**
List Processing. Eine Programmiersprache für die Verarbeitung von Listenstrukturen und rekursiver Technik für Probleme der künstlichen Intelligenz. Beispiel:

```
(DEFLIST
 ((CAAR(LAMBDA(X)(CAR(CAR X))))
  (CADR(LAMBDA(X)(CAR(CDR X))))
  (CDAR(LAMBDA(X)(CDR(CAR X))))
  (CDDR(LAMBDA(X)(CDR(CDR X)))) )
 EXPR )
```

**Listing**
Ausdruck, Auflistung

**Loader**
Ein Ladeprogramm
**Logik Analysator**
Ein Geräte, mit dem man digitale Schaltungen auf ihr Zeitverhalten untersuchen kann. Ferner gibt es für Mikroprozessoren auch die Möglichkeit den Befehlsablauf sichtbar zu machen.
**Logo**
Eine Programmiersprache, die ähnlich wie Lisp auch mit Listen arbeitet, jedoch zusätzlich graphische Ausgabebefehle besitzt. Die Sprache wurde zum Programmieren und Lernen für Kinder entwickelt, bietet jedoch Fähigkeiten, die bis in den Hochschulbereich reichen. Die Sprache gibt es für unterschiedliche Nationalitäten. Eine Besonderheit ist, daß man sich selbst Befehle definieren kann, die dann die Sprache erweitern.
Beispiel:

```
LERNE #ÜBERSETZE :L
 WENN :L = [ ] DANN RÜCKKEHR
  DEF ERSTES :L #UEB PRLISTE
                     ERSTES :L
 #ÜBERSETZE OHNEERSTES :L
ENDE
LERNE KREIS :N
 WH 360 [ VW :N RE 1]
ENDE
```

**Loop**
Schleife; durch einen Sprung zurück kann eine Programmschleife entstehen.
**Low order**
Niederwertige Stelle

## M

**Mark**
Bei der seriellen Übertragungs ist damit der logische Wert 1 gemeint, im Gegensatz zu Space.
**Maschinenbefehl**
Ein Befehl, den der Computer unmittelbar verstehen kann.
**maskieren**
Damit kann die Ausführung von Interrupts z.B. gestoppt oder freigegeben werden. Ferner bedeutet maskieren auch bestimmte Bits ausblenden.
**Memory**
Speicher
**Mikrocomputer**
Besteht aus einem Mikroprozessor, Speichern und Peripherie.
**Mikroprogrammierbar**
Der Befehlssatz eines Prozessors kann mit Hilfe von Mikrobefehlen definiert werden. Nicht mit Maschinensprache zu verwechseln. Der 68000/8 enthält z.B. ein Mikroprogramm das in seinem Inneren abgelegt ist und den Befehlssatz definiert, es kann jedoch nicht verändert werden.
**Mikroprozessor**
Ein integrierter Baustein, als Teil eines Mikrocomputers, der ein Leit- und Rechenwerk besitzt.
**Mikrorechner**
Ein Computer, der mit einem Mikroprozessor aufgebaut ist.
**mnemotechnische Darstellung**
Leicht zu merkende Abkürzungen für längere Begriffe, z.B. ist BRA die mnemotechnische Abkürzung für Branch.
**Modem**
Modulator und Demodulator. Eine Schaltung, die Daten für eine Fernübertragung aufbereitet. Ein Akkustikkoppler ist zum Beispiel ein solches Modem.
**Modula 2**
Eine Programmiersprache, die alle Konzepte von Pascal enthält, zusätzlich jedoch das Modul-Konzept beinhaltet.
Beispiel:

```
DEFINITION MODULE Buffer;
  EXPORT QUALIFIED ablegen, holen,
    nichtleer, nichtvoll;
  VAR nichtleer, nichtvoll : BOOLEAN;
  PROCEDURE ablegen(x : CARDINAL);
  PROCEDURE abholen(VAR x :
                         CARDINAL);
 END Buffer.
```

**Monoflop**
Ein bistabiles Speicherelement, das nach dem Auslösen nur eine bestimmte Zeit in dem neuen Zustand bleibt und danach wieder in den Ruhezustand zurückfällt.
**Multiplex**
Übertragung von mehreren verschiedenen Informationen, die dazu zeitlich hintereinander übertragen werden.
**Multiprozessing**
Ein aus mehreren CPUs oder Teil-Computern zusammengesetzter Rechner.

## N

**Nesting**
Verschachtelung; z.B. verschachteln von Unterprogrammen.
**NMI**
Non Maskable Interrupt. Eine Unterbrechung, die nicht gesperrt werden kann.

## O

**Odd**
bedeutet ungerade. Die Zahl 3 ist zum Beispiel ungerade.

**offener Kollektor**
Schaltung, dessen Endtransistor einen herausgeführten, unbeschalteten Kollektor hat.
**Oktal**
Zahlendarstellung zur Basis 8
**Output**
Ausgabe

## P

**Packen**
Dabei werden z.B. zwei Dezimalzahlen in einem Byte untergebracht
**Parity**
Parität, Gleichheit
**Pascal**
Eine höhere Programmiersprache, die für Lehrzwecke entworfen wurde und zunehmend Verbreitung findet. Siehe Kapitel Pascal/S
**Pass**
Lauf oder auch Durchgang, z.B. bei einem Übersetzungsvorgang
**PE-Verfahren**
Phase encoding. Verfahren bei der Aufzeichnung auf Datenträger, siehe Kapitel CAS
**Pegel**
Spannungsbereich
**Peripherie-Geräte**
Einheiten, die mit der Außenwelt eines Computer in Verbindung treten können.
**Pipelining**
Fließbandverarbeitung. An mehreren Stellen wird in kleinen Schritten gleichzeitig eine Verarbeitung vorgenommen. Dadurch kann man in Mikroprozessoren die Befehlszykluszeiten verkleinern.
**PL/1**
Programming Language 1. Eine höhere Programmiersprache, die zur Pascal-Familie gehört. Beispiel:

```
TEST: PROCEDURE OPTIONS(MAIN);
 DECLARE (A,B) FIXED DECIMAL
  (6,2),
  (COUNT) FIXED;
 A = 12.34; B = A + 1.02;
 PUT SKIP(2);
 DO COUNT=1 TO 10;
   PUT EDIT('I=',I,'B=',B) (F(5),F(6,2));
   B = B + 1.0;
 END;
END TEST;
```

**PL/M**
Ähnlich der Programmiersprache PL/1, jedoch eine Teilmenge daraus. Wurde vorwiegend für die 8080-Prozessorfamilie verwendet.
**Plotter**
Ein Gerät, ähnlich zu einem XY-Schreiber, für die Ausgabe von graphischen Darstellungen.
**Pointer**
Zeiger. Ein Speicherplatz, der eine Adresse eines anderen Speicherplatzes enthält.
**Polling**
Aufrufbetrieb. Darunter versteht man die ständige Abfrage, z.B. eines Peripheriegerätes, um so festzustellen, ob es schon fertig ist.
**Port**
Tor. Man meint damit Ein- oder Ausgabebausteine
**Prellen**
Mechanische Schalter berühren die Kontaktflächen beim Schließen oder Öffnen mehrere Male.
**Programm**
Ist eine Folge von Anweisungen (Befehlen), die zur Lösung eines Problems dienen sollen.
**Programmiersprache**
Eine Sprache zur Formulierung von Programmen, die automatisch (von einem Übersetzungsprogramm oder Interpreter) in Maschinensprache umgesetzt werden können.
**Programmspeicher**
Dort ist das auszuführende Programm abgelegt.
**Programmzähler**
Er legt die Adresse der Speicherzelle des nächsten Befehls fest.
**PROM**
Programmable Read Only Memory. Ein Festwertspeicher, der durch Anlegen elektrischer Impulse beschrieben werden kann.
**Pseudobefehl**
Eine Instruktion, die nicht im Befehlssatz des Prozessors vorhanden ist, und zur Steuerung des Assemblers dient.
**Pull-Up-Widerstand**
Ein Widerstand, nach +5V geschaltet, um z.B. eine offene Kollektorschaltung zu beschalten.

## Q

**Queue**
Warteschlange. Daten werden in einer Warteschlage angesammelt, wenn sie noch nicht verarbeitet sind.

## R

**RAM**
Random Access Memory. Speicher mit wahlfreiem Zugriff
**Real time clock**
Echtzeituhr
**Redundanz**
Teil einer Nachricht, die zur eigentlichen Information nichts mehr beiträgt.
**refresh**
Wiederauffrischen

**Relokalisierbar**
Ein Programm, das in verschiedenen Speicherbereichen lauffähig ist.
**Reset**
Rücksetzen
**ROM**
Read Only Memory; ein Speicher, den man nur auslesen kann.

## S

**scan**
Abtasten
**scrollen**
Der Bildschirminhalt wird nach oben oder unten verschoben.
**sedezimal**
Zahlendarstellung zur Basis 16
**select**
Auswählen
**sense**
Abtasten
**Simulator**
Ein Programm, das einen Vorgang künstlich nachbildet.
**Software**
Hierunter versteht man alle Arten von Programmen, wie auch Texte und Informationen.
**Source**
Quelle
**Space**
Bei der seriellen Übertragung ist damit der logische Wert 0 gemeint.
**Space-Taste**
Leertaste auf der Tastatur, die einen Freiraum erzeugt.
**Speicherbaustein**
Ein Baustein, der Informationen behalten kann.
**Stack**
Stapelspeicher, Kellerspeicher. Siehe LIFO
**State**
Zustand
**Statement**
Anweisung, Befehl
**statische RAMs**
Speicher, die z.B. mit zwei Transistorzellen aufgebaut sind und wie ein bistabiles Flip-Flop arbeiten.
**Steuerwerk**
Dieser Teil des Computers kontrolliert die Ausführung sämtlicher Befehle, er wird auch mit Leitwerk bezeichnet.
**Strukturierte Programmierung**
Verfahrensweise, um einfach zu testende und verständliche Programme zu erzeugen.
**Subroutine**
Unterprogramm
**Supervisor**
Ein Organisationsprogramm

## T

**Tantal-Kondensator**
Wird in Mikroprozessorschaltungen gerne zur Unterdrückung von Spannungsspitzen auf der Versorgungsleitung verwendet.
**Terminal**
Datenendstation
**Text-Editor**
Siehe Editor
**Time sharing**
Zeitscheibenverfahren. Dabei können mehrere Benutzer auf ein und denselben Computer zugreifen.
**Timing-Diagramm**
Zeitlicher Ablauf, bildlich dargestellt.
**Trace**
Ablaufverfolgung; Methode zur Fehlersuche in Programmen.
**transfer**
Übertragen
**Tristate-Treiber**
Ein Schaltkreis, der drei Zustände besitzt. Pegel auf 0, Pegel auf 1 oder offen (Pegel undefiniert).

## U

**UART**
Universal Asynchronous Receiver/ Transmitter.
Die Schaltung dient der seriellen Übertragung.
**Unit**
Einheit, Gerät
**Unterprogramm**
Gleiche Befehlsfolgen, die in einem Programm mehrfach vorkommen, kann man zu Unterprogrammen zusammenfassen, und muß sie daher nur einmal abspeichern.

## V

**V24**
Schnittstellen-Norm für serielle Signale.
**Valid**
gültig
**Vektor Interrupt**
Das auslösende Gerät gibt zusätzlich zur Interrupt-Anforderung auch noch eine Zieladresse vor.

## W

**Wait**
Warten
**Wired-Or**
Eine logische Verknüpfung, die nur durch die Verdrahtung entsteht.
**Worst case**
Ungünstigster Fall
**Wort**
Zusammenfassung mehrerer Bits zu einer logischen Einheit.

## Z

**Z80-CPU**
Ein Mikroprozessor-Baustein
**Zeichengenerator**
Der Zeichensatz für die Schriftdarstellung auf dem Bildschirm ist darin gespeichert.
**Zugriff**
Zugang z.B. in eine bestimmte Speicherzelle
**Zyklus**
Eine Anzahl von Schritten, die wiederholt werden und im Ablauf gewisse Ähnlichkeiten aufweisen

# Sachverzeichnis

Klein
Amiga: Programmieren in Basic

Der Amiga macht besonders im Bereich der Grafik deutlich, was unter einem modernen und leistungsfähigen Rechner zu verstehen ist.

Windows, die Maus, Ton- und Sprachausgabe, sowie eine gigantische Rechenleistung fordern den Programmierer heraus. Das ewig junge Basic präsentiert sich im Amiga in einem völlig neuen Kleid. Interessante neue Befehle, der Wegfall der Zeilennummern, die lokalen Variablen, der Unterprogramm-Aufruf per Name oder die Aufteilung in Basic- und Listfenster zeigen den Charakter von Amiga-Basic.

Der Autor nimmt den eingefleischten Basic-Könner ebenso wie den Einsteiger an die Hand und führt ihn in diese faszinierende Basic-Welt. Ohne verstaubte und langatmige Theorie geht es sofort ran an die Tastatur. Die im Buch liegende Diskette spart dabei das zeitaufwendige und fehlerträchtige Listing-Abtippen.

Die Beherrschung der vollen Amiga-Leistung ist das hochgesteckte Ziel des Buches. Es wird ohne unnötigen Ballast und auf sicherem Wege erreicht.

Rolf-Dieter Klein ist durch zahlreiche Buch- und Zeitschriften-Publikationen ebenso bekannt, wie durch seine Fernsehsendungen. Er ist der Vater des weitverbreiteten NDR-Klein-Computers.

04800
9 783772 389719
ISB N 3-7723-8971-6 DM +048.00

Franzis'